Packing curved objects with interval methods

Ignacio Antonio Salas Donoso

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

MINES Nantes
Institut Mines-Télécom

Thèse de Doctorat

Ignacio Antonio SALAS DONOSO

Mémoire présenté en vue de l'obtention du

grade de Docteur de l'École nationale supérieure des mines de Nantes

sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications

Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 29 avril 2016

Thèse n° : 2016EMNA277

Packing Curved Objects with Interval Methods

JURY

Rapporteurs :	**M. Luc JAULIN**, Professeur des universités, ENSTA Bretagne
	M. Gilles TROMBETTONI, Professeur des universités, Université de Montpellier
Examinateur :	**M. François FAGES**, Directeur de recherche, INRIA Saclay
Directeur de thèse :	**M. Nicolas BELDICEANU**, Professeur des grandes écoles, Mines Nantes
Co-encadrant de thèse :	**M. Gilles CHABERT**, Maître assistant, Mines Nantes

Acknowledgments

I would like to thank my advisor, Dr. Gilles Chabert, for his support and patience to me. Without him would be impossible to finish this thesis.

I am grateful to Dr. Maria Cristina Riff for believe in me and encourage me to cross the Atlantic and make front to new opportunities.

I would like to thank to my family, my parents, Juan Salas and Sylvia Donoso, my brother Nicolas Salas and my uncle Julio Donoso, they give me from the very beginning all his support, and his full confidence allows me to go beyond. Without them I never could have come here.

I am particularly grateful with my sweet Amanda, my girlfriend and future wife, who became in an emotional pillar that helps me to continue, even when the motivation goes down.

Finally I am grateful to all those other persons that I have met here and became a family outside my country: Jesus Rodriguez, Agustina Campesi, Gustavo Soto, Belen Rolandi, Jonathan Pastor and Gergana Mitrusheva.

1

Introduction

1.1 Goal of the Thesis

1.1.1 Context

A common problem in logistic [Zachariadis 2012], warehousing [Jouida 2015], industrial manufacture [Cui 2013], newspaper paging [Lodi 2002] or energy management in data centers [Cambazard 2013], is to allocate items in a given enclosing space or *container*. This is called a *packing problem*.

Packing problems are divided into classes depending on the shape of items. Well-known classes include the *bin packing* [Lodi 2002, Madhumathi 2015] (see Fig. 1.1), the *circle packing* [Hifi 2009, Pospichal 2015], the *knapsack problem* [Pisinger 2007, Lin 2015], the *layout problem* [Drira 2007, Tari 2015] or the *cutting problem* [Cheng 1994]. These classes have taken root in industrial applications or academic challenges.

Business-specific constraints usually appear in addition to the basic geometrical problem. One example is the requirement for an item to be close to another, or for some objects to respect a more complex arrangement. This happens, e.g., in the manufacturing industry, where the items to pack are *facilities* that form part of a production chain [Drira 2007]. In this context, a correct arrangement of the facilities minimizes the production cost. These specific constraints are usually integrated on top of a general-purpose algorithm designed for a given class of problems, like the *bin packing*.

The goal of this thesis is to allow arbitrary shapes, as long as they can be described mathematically (by an algebraic equation or a parametric function). In particular, shapes can be curved and non-convex. This is what we call the *generic packing problem*.

The aforementioned classes consider fixed shapes like boxes, polygons or circles and, of course, the al-

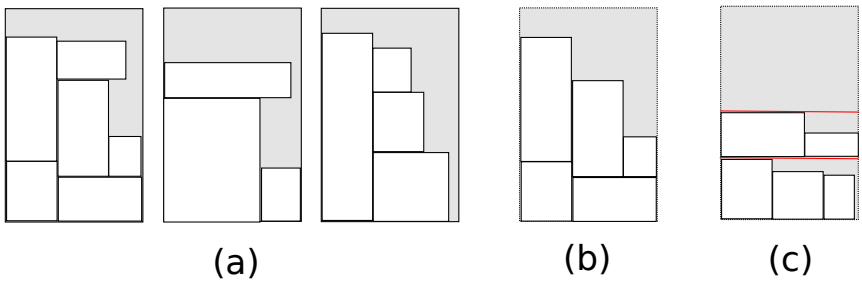


Figure 1.1 – **Bin packing and some of its variants.** Each figure depicts a solution. **(a)** Bin packing: the number of bins is minimized (there are 3 bins). **(b)** *Strip packing*: the height of the bin is minimized. **(c)** *Guillotine packing*: the number of levels (lines in red) is minimized.

gorithms in the literature are closely dependent to the considered shape. But we can say that these problems are actually not on the same complexity level that the generic one.

For instance, the case of boxes allow to cast the continuous packing problem into a discrete one since it is easy to define a grid where vertices of rectangles have to be anchored.

The other shapes require continuous reasoning but their simplicity usually allow to formalize the packing problem as explicit constraints on the item positions. As an example, two circles do not overlap if and only if the distance between their centers is greater than the sum of their radii. So, the circle packing problem can be seen as a regular continuous constraint satisfaction problem. The same holds with polygon packing, albeit less obviously.

The generic packing problem is of higher complexity and still represents an open field of research. Many works, especially in CAD, resort to a polygonal discretization of objects (see §1.3) and deal with polygon packing. However, the resulting polygon may not rigorously enclose the object, which may lead to an unsafe packing solution, and polygon packing is still hard to handle when rotation is allowed.

1.1.2 A Decision Problem

Independent of the nature of items, the packing problem usually comes into three variants:

1. the items and the container are given,
2. the container is given and the number of items has to be maximized
3. the items are given and the size of the container has to be minimized.

It may happen that the container does not appear in the problem formulation: e.g., one can place items so as to minimize the surface (or volume) covered by the items. But, in this case, the hull of the surface can simply be interpreted as a container of variable shape, whose size has to be minimized so that this type of problem falls again in our third category. It may also happen that the container has a mixed discrete-continuous nature. A very concrete example is when we have to minimize the number of bins used for shipping a given quantity of goods. In this case, the container actually represents the set of bins, as a whole. Therefore, the description of the container has continuous components (height, width and depth of the bin) and a discrete one (number of bins). Note that only the discrete component has to be minimized in this case.

This thesis will only focus on the first type of problems, where both the container and the items are

fixed. Since there is no optimization step, the packing problem is supposed to involve placing items in an organized and non-trivial way. As a well-known complexity principle, an instance of a combinatorial problem with many solutions or, at the opposite, a very few (or none) are two extreme cases where solving becomes easy. A complexity peak lies somewhere between both, where the solver wastes computational resources exploring non-feasible solutions without finding a feasible one. This situation in the packing problem corresponds to instances with a tight space, but still with tiny movements possible inside. So, although not explicitly stated, our packing problem implies a notion of *compactness*, that is, a minimal loss of space.

1.1.3 Summary

The goal of this PhD is to develop a numerically guaranteed algorithm for packing a set of items that can be both translated and rotated in a given container, all being of arbitrary shapes, without resorting to an a priori discretization. This is what the *generic packing problem* will now refer to.

1.2 Definitions

This section introduces the definitions and key concepts for formalizing the problem.

1.2.1 Object Definition

For clarity, let us first assume that the orientation is fixed, that is, objects can only be translated.

Translating an object means fixing the position of a particular point that we call the *origin*. This origin point can be arbitrarily chosen. For instance, in rectangle packing, the origin of a rectangle can be a vertex or its center point. Once this convention for the origin made, the shape of the object is just a regular constraint. To illustrate this, let us consider again rectangle packing. If the origin is the lower-left corner (see Figure 1.2.a), then a rectangle of dimensions l_1 and l_2 is the set of all $p \in \mathbb{R}^2$ satisfying

$$c(p) \iff 0 \leq p_1 \leq l_1 \wedge 0 \leq p_2 \leq l_2. \tag{1.1}$$

Alternatively,

$$c(p) \iff -\frac{l_1}{2} \leq p_1 \leq \frac{l_1}{2} \wedge -\frac{l_2}{2} \leq p_2 \leq \frac{l_2}{2} \tag{1.2}$$

defines a rectangle with the center point as origin (see Figure 1.2.b), which, of course, is just a shift of the previous constraint. So, the shape of an object can be expressed as a constraint, the latter containing an implicit convention for the origin. As a last example, a circle of radius r is usually defined as the set of all $p \in \mathbb{R}^2$ such that

$$c(p) \iff \|p\| \leq r, \tag{1.3}$$

the origin being, in this case, the center of the circle.

This definition of $c(p)$ corresponds to an object with no translation nor rotation. The general constraint corresponding to an object translated by x and rotated by an angle α is easy to obtain.

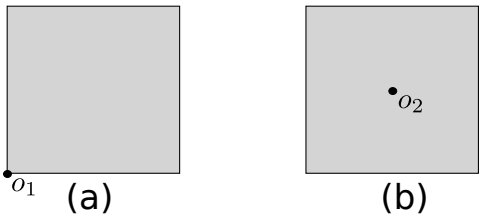


Figure 1.2 – **Two possible definitions for a square shape. (a)** The origin point o_1 is fixed in one corner of the square. **(b)** The origin point o_2 is fixed in the middle of the square.

However, before that, we need to say exactly what an object *translated by* x and *rotated by* α is, and there are actually several options, basically depending on the order transformations are made and in the choice of the origin point for rotation.

We can:

1. first, translate the object by x and then rotate the object by α around its new origin
2. first, translate the object by x and then rotate the object by α around the origin of the frame
3. first, rotate the object by α around its origin and then translate the object by x
4. etc.

In this thesis, we chose the first option and, clearly, this choice has a strong impact in the algorithms and formulas. One could wonder then if another convention would have given better results and this has not been investigated, having assumed that all options should be equivalent, in the whole.

So the first transformation we apply is translation. The part of the plane covered by an object translated by a vector (or placed at some position) x is the set of all p such that $c(p - x)$ is satisfied. We now apply rotation. By a classical geometric argument, the object placed at x and turned by some angle α is the constraint

$$c\big(R_{-\alpha}(p - x)\big) \tag{1.4}$$

where R_α is the rotation matrix with angle α:

$$R_\alpha = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}. \tag{1.5}$$

Equivalently, we can say that $c(p) \equiv c(R_0(p - 0))$ represents the object placed at 0 and rotated by the angle 0. To conclude:

— *an object* is a constraint on the plane (i.e., with two variables),

— *placing* an object means fixing the coordinate of its implicit origin,

— *orienting* an object means fixing the angle of the rotation around its implicit origin.

In this thesis, we first consider objects described by nonlinear inequalities $c(p) \iff f(p) \leq 0$. E.g., a circle of radius 1 which origin is the center point is the set of all $p \in \mathbb{R}^2$ such that $f(p) \leq 0$ with $f : p \mapsto \|p\| - 1$. In a second part, we consider objects described by parametric curves and, in this case, the constraint c is more complicated to handle.

For the clarity of presentation, we will often assume an object to be described by a single constraint (one inequality or one chain of curves), but our results have been easily extended to the more general case of objects described by first-order formulas, that is, disjunctions of conjunctions of constraints. In particular, there is no convexity assumption on the input objects.

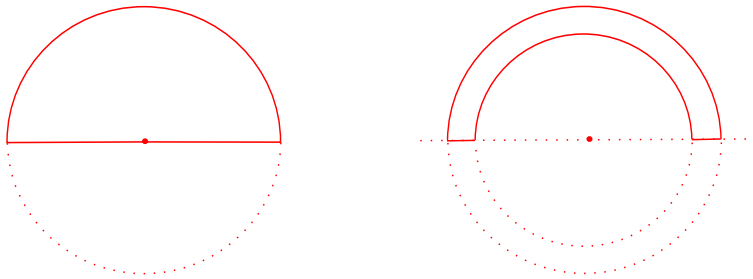


Figure 1.3 – **Half-circle and horseshoe.** The constraint for the half-circle of radius 1 is $c(p) \iff (\|p\| \leq 1 \wedge y_p \geq 0)$. The constraint for the horseshoe is $c(p) \iff (\|p\| \leq 1 \wedge \|p\| \geq 0.75 \wedge y_p \geq 0)$.

1.2.2 Object Configuration

Now that we know how an object can be translated or rotated, we can now focus on the non-overlapping constraint. But, before that, we introduce for commodity the notion of *configuration*, which gathers all the information related to one object (position and orientation).

This configuration is a vector of *parameters* that contains the position $o \in \mathbb{R}^d$ of an item, where d represents the geometric dimension (which is less or equal to 3) and its orientation, an angle $\alpha \in [0, 2\pi]$.

A solution of the packing problem is therefore the *configurations* of every items.

Definition 1 (Object configuration)
If Object i is placed at a position o_i and rotated by an angle α_i, we will denote by q_i and call the configuration of Object i the vector:

$$q_i := (o_i, \alpha_i) = (x_i, y_i, \alpha_i). \tag{1.6}$$

The size of q, which is the number of degrees of freedom, will be denoted by d. If only translation is considered, the angle is dropped and $d = 2$. Otherwise, $d = 3$.

1.2.3 Non-Overlapping Constraint

Let us now consider two objects of indices i and j. According to (1.4), Objects i and j overlap iff

$$overlap_{ij}(q_i, q_j) \iff \exists p \in \mathbb{R}^2, \quad c_i(R_{-\alpha_i}(p - o_i)) \wedge c_j(R_{-\alpha_j}(p - o_j)). \tag{1.7}$$

This is depicted in Figure 1.4.

The non-overlapping constraint between two objects is just the negation of the latter relation.

Definition 2 (Overlapping Constraint)
Two objects i and j of shapes c_i and c_j respectively, which configurations are $q_i = (o_i, \alpha_i)$ and $q_j = (o_j, \alpha_j)$, satisfy the overlapping constraint

$$overlap_{ij}(q_i, q_j)$$

iff

$$\exists p \in \mathbb{R}^2, \quad c_i(R_{-\alpha_i}(p - o_i)) \wedge c_j(R_{-\alpha_j}(p - o_j)). \tag{1.8}$$

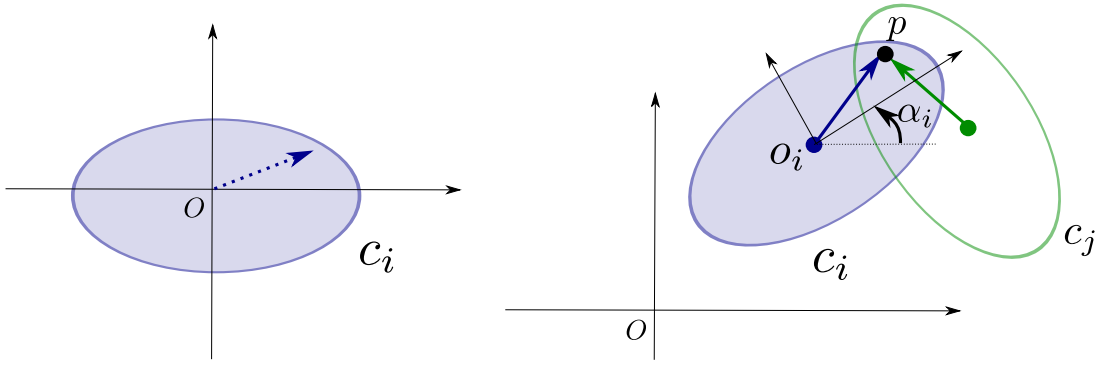


Figure 1.4 – **The overlapping constraint.**

1.2.4 Problem to be Solved

The packing problem is a set of pairwise non-overlapping constraints between n objects and an additional constraint that all objects must be packed inside some *container* (like the enclosing bin in the bin packing). Let us call the *inclusion constraint* this additional constraint. Note that the container, as the other objects, can have an arbitrary shape.

In most of specific packing problems, the container space provides a guidance to place the objects in order to maintain compactness. In our case, the shapes are more abstract and there will be no such heuristic; in fact, as we show below, the container can be considered as an item among others and the only constraint we will have to deal with is actually the non-overlapping constraint.

1.2.5 Handling the Container

One appealing aspect of our representation is that we don't actually need a specific treatment for the inclusion constraint. This is based on a very simple property. An object is inside the container iff this object does not overlap the complementary of the container. And the complementary of a shape is simply obtained with the negation of the underlying constraint. So, in our system, the packing problem consisting in placing n objects of shapes $c_1, \ldots, c_n$ inside a container of shape c is reformulated as non-overlapping constraints between $n+1$ shapes $c_1, \ldots, c_n, \neg c$.

Finally, the problem to be solved is a system of $\binom{n+1}{2}$ pairwise non-overlapping constraints between all pairs of objects, including the complementary of the container. This is illustrated in Figure 1.5.

Note that, for obvious symmetry breaking reasons, one has to fix the origin and the orientation of one object. A natural choice is then to fix that of the container.

1.3 State of the Art

There already exist in the literature several packing frameworks that are not dedicated to a specific shape and where a great freedom in the object definition is already proposed. These frameworks are not as general

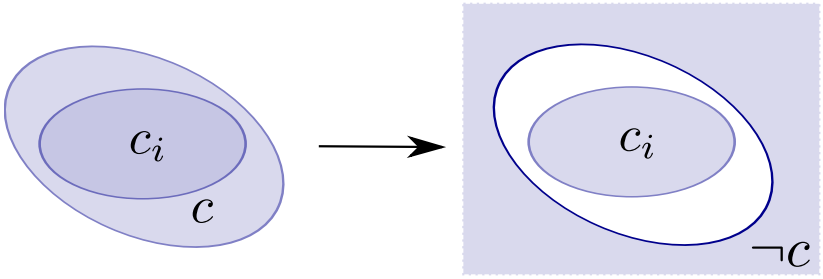


Figure 1.5 – **The inclusion constraint as a non-overlapping constraint.** The complementary of the container is just an object among the others.

as ours but they share many similarities. Three of them are described in §1.3.1.

Other well-known problems that have strong connections with generic packing are introduced in §1.3.2.

1.3.1 Generic Packing

Generic packing frameworks usually split the problem in three parts. First, a violation function f_{ij} for each pair (i, j) of objects is built. This function takes as argument the position $o = (x, y)$ and the orientation α for both objects and gives a measure of "how much" the two objects overlap. We shall call later such function an *overlapping function* (in Chapter 3).

Second, all these violations are summed up to form a global violation cost f:

$$f(o_1, \alpha_1, \ldots, o_n, \alpha_n) := \sum_{i>j} f_{ij}(o_i, \alpha_i, o_j, \alpha_j).$$

Third, the function f is minimized using various optimization techniques.

Some of the experimental results found in the literature are very encouraging. However, there is a serious bottleneck in these approaches: either the objects are approximated by polygons or the overlapping functions are ad-hoc formulas for every possible combination of object shapes. For instance, in a problem where objects are either circles, squares or triangles, one must build a formula for the 6 combinations: *circle-circle, circle-square, circle-triangle, square-square, square-triangle* and *triangle-triangle*. Some formulas are easy to obtain, like for the *circle-circle* case. Indeed, given two circles of radii r_i and r_j, the overlapping can be defined as the distance between the two centers and 0 if this distance exceeds the sum of the radii, that is:

$$f_{ij}(o_i, \alpha_i, o_j, \alpha_j) := \max(0, r_i + r_j - \|o_i - o_j\|).$$

This simple formula is rather an exception. Mixing shapes considerably complicate the task – even the *circle-rectangle* case is not so simple–. One should be convinced that, in general, these ad-hoc formulas are difficult to write. And their number grows in $\mathcal{O}(n^2)$.

Our goal will be precisely to replace these formulas by an algorithm.

Phi-Objects

One of such framework has been proposed in [Chernov 2010, Stoyan 2013].

The objects that can be built in this framework are called *phi-objects*. They are built by assembling basic shapes called *primary phi-objects*. The primary phi-objects in $\mathbb{R}^2$ are circles, rectangles, regular polygons or convex polygons.

Then, more complex objects or *composed phi-objects* are obtained with conjunctions and disjunctions of primary phi-objects, like we did in §1.2.1. Composed phi-objects can be composed in turn, leading to a multi-level construction scheme.

One interesting aspect of this framework is that the boundary and the interior of an object are considered separately, which allows to make a distinction between a tangent and a plain intersection. This however requires that the composition of objects does not create self-intersections. Furthermore, since the closure of the complement of a phi-object is also a phi-object, the same transformation for the container as in §1.2.5 can be applied in their context to rule out the inclusion constraint.

The function they minimize is not exactly an aggregation of *overlapping functions* but so-called *phi-functions* that give a measure of the distance between two objects. More precisely, a phi-function returns a negative value for disjoint object, a positive value for plain-intersecting objects, and zero in case of tangency. We will see that, in our case, the distance between objects is replaced by a measure of violation, so that the returned value is zero for disjoint object and positive otherwise.

As said above, the phi-function for two objects is an ad-hoc formula, based on the geometrical properties of the primary phi-objects. However, the formula may become complex for a composed phi-object and, moreover, involve radicals that hinders differentiation. Since they resort to a gradient-based optimization, an exact formula may thus be inadequate. So the phi-function is often replaced by a simpler formula, that just roughly estimates the real distance between the objects.

One advantage of this framework is that packing is also possible in three dimensions. The primary phi-objects in $\mathbb{R}^3$ are spheres, parallelepipeds, cylinders, cones or convex polytopes.

Packing with CMA-ES

Another framework closely related to ours is the one proposed in [Martinez 2013]. In fact, this paper inspired this thesis for a large part.

The minimization is performed this time by an evolutionary algorithm called CMA-ES [Hansen 2001] which is also the algorithm we have used in our experiments (see §2.8 for details).

One specificity of their approach is that they don't explicitly consider a container but they rather try to pack objects by minimizing the overall space.

So, their objective function f is a combination of two factors f_o and f_s. The function f_o is a sum of functions that exactly match our definition of *overlapping functions*; however, they did not really formalized it this way. The function f_s is a measure of compactness. Typically, the function f_s corresponds to the minimal surface of a rectangle that encloses all the objects. Another possible alternative is the minimal radius of an enclosing circle.

Finally, f is a linear combination of both, namely, $f = \beta f_o + f_s$, where β is a positive coefficient. When β is very large, the violation of the non-overlapping constraint is strongly penalized, which makes this violation the first factor to be minimized. When the non-overlapping constraint is satisfied, or nearly

satisfied, then the packing surface is minimized.

Polygon and Medial Axis Approximations

A similar framework is also developed in the thesis [Jacquenot 2010]. The originality of this framework lies in a mix of quasi-Newton and genetic algorithm for minimization, and in the fact that the calculation of the overlapping function is performed differently whether the angle α takes discrete or continuous values.

In the discrete case, objects are approximated by polygons. The overlapping function is then calculated from the *No-Fit Polygon* [Gomes 2006, Domovic 2014]. This polygon describes, for given orientations of the objects, all the positions for one of them such that the two overlap (this concept will be taken up and extended to the concept of *overlapping region* in §3.1.3). Similarly, the inclusion constraint of an object in the container is based on a similar polygon called *Inner-Fit Polygon* [Sato 2012]. The Inner-Fit Polygon describes, given fixed orientations, all the positions for the object that makes it belong to the container. Finally, the overlapping function is evaluated by considering all these no-fit and inner-fit polygons for the different possible values of α.

In the continuous case, the latter techniques is not adapted due to the infinite number of possible values for the angle. Objects are therefore represented differently, namely, by an assembly of circles. This representation is called *Medial Axis* [Tănase 2004]. The medial axis is basically the set of the centers of inner disks that touch the boundary of the object tangentially in several points. A dedicated algorithm for evaluating the overlapping function with this representation is proposed, based on the *circle-circle* distance formula that we already mentioned above.

Then, a safe packing is obtained by applying a quasi-Newton BFGS method [Broyden 1970, Zhang 2013] on the overlapping function, which locally translates or rotates the objects. This is called the *separation algorithm*.

In the context of this PhD, not only a solution of the packing problem is sought but a solution that minimizes some other criterion. The minimization of this criterion is therefore performed by embedding the separation algorithm inside a multi-objective genetic algorithm called Omni-Optimizer [Deb 2008] that introduces mutations and crossover operators over the population to create diversity in the search, that is, to generate global movements. The initial population can be constructed randomly or can be directly given by the user. The multi-objective approach also allows to integrate specific constraints in addition to the geometrical packing problem.

1.3.2 Other Related Works

Jigsaw Puzzles

A different problem but related to our packing problem is the *jigsaw puzzle problem* with polygons [Wolfson 1988]. This problem also consists in packing n pieces inside a container but, unlike the packing problem, the solution entirely fills the space and two contiguous pieces have parts of their boundaries that perfectly fit.

A method for solving jigsaw puzzles is presented in [Mistry 2014] that exploits so-called *cavities* and

protrusions of 2D polygons.

Given a polygon $Q \subset \mathbb{R}^2$ with a convex hull H, a cavity of Q is the closure of an open connected region of $H \setminus Q$. The regions of Q between two cavities are called protrusions.

A puzzle is then solved by considering the following steps:
— cavity-protrusion hierarchy computation,
— pairwise matching and
— global matching and orientation computation.

The *hierarchy* of an initial polygon is a n-ary tree, where each node represents a subdivision of the initial polygon and each leaf represents either a cavity or a protrusion polygon produced by this subdivision. The hierarchy of cavities and protrusions is built for every polygon and used to check if the protrusion of a polygon fits inside a cavity of another one.

More precisely, the cavity-protrusion matching is performed by comparing two hierarchies with a backtracking scheme. Given two hierarchies H_1 and H_2 describing two polygons, cavity or protrusion leaves of H_1 are compared to the nodes of the H_2. This node comparison is, in fact, a comparison between polygons. If the surface of the leaf polygon of H_1 is larger than the surface of the polygon of the current node of H_2, the whole current branch of H_2 is rejected. Also, when the leaf polygon of H_1 and the polygon of the current node of H_2 are of the same type (protrusion or cavity), the branch is also rejected. Finally, if two leaves match, that is, if there exists a cavity-protrusion complementarity and if the area of the protrusion is lower or equal to that of the cavity then a finer edge-based comparison between the cavity/protrusion polygons is made to verify if the protrusion may fit inside the cavity.

So this data structure allows the early rejection of mismatches and is able to detect unique cavity-protrusion matches.

Note that this structure cannot be adapted in our context of arbitrary curved objects.

Collision Detection

The non-overlapping constraint has a close relation with the *collision detection* problem [Cohen 1995, Brochu 2012, Mainzer 2015]. This problem is fundamental in computer animation [Moore 1988], physically-based modeling, computer simulated environments and robotics. The objective of collision detection is to report all geometric contacts between objects, where some of them can be in motion.

The PhD thesis [Lin 1993] presents a survey of collision detection algorithms for different types of objects including curved objects. Like us, they consider the case where the objects are described by parametric functions (splines models) or inequalities. Their main objective is to verify if two objects are in collision and, otherwise, to find a pair of points (one for each object) that minimizes the distance between the objects.

The collision detection is based on the *contact conditions*, that we shall describe in more details for the 2D-case in §4.3. Since these conditions are based on differential calculus, different types of collision are considered, depending if the two objects are tangent or if one object touches the other at an edge or a vertex.

Collision detection algorithm also exist for parametric or algebraic surfaces.

Quantified Constraints

Definition 2 also means that our problem falls into the category of existentially-quantified constraints.

A state-of-the-art algorithm for calculating a paving with existentially-quantified inequalities is given in [Ratschan 2006a] and is implemented in the RSOLVER tool [Ratschan 2006b]. Rsolver implements a general algorithm, somehow a numerical version of CAD, for solving quantified constraints.

General techniques [Goldsztejn 2006, Ishii 2012] for quantified *equality* constraints could also be used: either by adapting [Goldsztejn 2006] to compute an over approximation of the boundary of the overlapping region (see Chapter 3), or using a necessary condition for the boundary of the overlapping region expressed as an under-constrained system of equations and using [Ishii 2012]. However, the overlapping constraint naturally involves inequality constraints and using costly techniques dedicated to equality constraints is counterproductive. This will be discussed in Section 4.3.

1.4 Overview of the Thesis

The manuscript is organized as follows.

In Chapter 2 is given the background required to build our framework for solving the generic packing problem. This background belongs for the main part to interval analysis. We shall make a quick survey of this field, mentioning only the notions and techniques that are necessary for the understanding of the rest of the document.

Chapter 3 gives the main ingredients of our packing framework. The first ingredient is a modelization of the packing problem as a minimization problem (that we solve with an evolutionary algorithm), which represents a global violation cost. The second ingredient is a decomposition of this cost as a sum of *overlapping functions* which measure the overlapping between two objects. The last ingredient is a numerical algorithm for evaluating an overlapping function, based on a pre-calculated set called *overlapping region* that represents the set of all relative configurations of one object (with respect to another) that creates an overlapping. This overlapping region is automatically calculated for each pair of objects.

The underlying algorithm for calculating the overlapping region depends whether objects are described by inequalities or parametric curves, like Bézier curves. Chapters 4 and 5 instantiate this framework for these two different classes of objects respectively by detailing how this region can be computed for each case. The computation requires, in particular, an *inflator*, an algorithm that builds a *box* which is proven to be inside the overlapping region. Our key idea is to split this inflation in two parts: an inflation with respect to translation and an inflation with respect to rotation. Experimental results are given at the end of both chapters.

It should be emphasized that a packing problem mixing these two classes of objects (inequalities and parametric curves) will not be covered in this document and would be a natural extension of the present work. This will be discussed in the conclusion.

2

Background

2.1 Introduction

This chapters presents some existing techniques this thesis is based on to solve the packing problem. It mainly consists in a survey of interval analysis and continuous constraint programming. These techniques are indeed at the core of our algorithm for describing the overlapping constraint. The chapter also quickly introduces the *CMA-ES* algorithm, that we used as a black-box optimization program.

2.2 Intervals and Interval Arithmetic

Interval arithmetic is an extension of the real arithmetic to the set of intervals.

A real interval is a connected subset of $\mathbb{R}$. An interval can therefore be either

$\emptyset$	The empty set
$(-\infty, +\infty)$	the real line
$(-\infty, b]$	a half-line with upper bound
$[a, +\infty)$	a half-line with lower bound
$[a, b]$	a segment

Intervals are denoted with brackets, e.g. $[x]$ and the set of intervals is $\mathbb{IR}$. The lower bound and upper bound are respectively denoted by $\underline{x}$ and $\overline{x}$, which are values of $\mathbb{R} \cup \{-\infty, \infty\}$, so that for any non-empty interval $[x]$ we have

$$[x] = [\underline{x}, \overline{x}].$$

The *width* of an interval is

$$w([x]) := \overline{x} - \underline{x}$$

and, by convention, 0 for the empty set.

The width is then equal to $+\infty$ in case of unbounded intervals. When $w([x])$ is equal to 0, the interval is said to be *degenerated*.

The center of a bounded interval, or midpoint, is

$$mid([x]) = \frac{\underline{x} + \overline{x}}{2}.$$

This definition assumes that $[x]$ is bounded. However, in algorithms, we often use the midpoint of an interval as a natural candidate when we just want to take a sample point inside the interior of the interval. So, in this situation, the midpoint of an unbounded interval also makes sense; one just has to fix some conventions. For example, in the IBEX documentation, the midpoint of $(-\infty, +\infty)$ is 0, which seems a reasonable choice (but choosing a convention for the midpoint of half-lines is less obvious).

Usual set operations can be easily performed with intervals by applying operations on the bounds.

For instance, the intersection of two intervals is obtained via $\min/\max$ operations on the bounds:

$$[x] \cap [y] = \begin{cases} [\max\{\underline{x}, \underline{y}\}, \min\{\overline{x}, \overline{y}\}], \text{ if } \max\{\underline{x}, \underline{y}\} \leq \min\{\overline{x}, \overline{y}\} \\ \emptyset \text{ otherwise} \end{cases}$$

The inclusion is obtained via comparison operations on the bounds:

$$[x] \subseteq [y] \iff \begin{cases} x \neq \emptyset \wedge y \neq \emptyset \wedge \underline{x} \geq \underline{y} \wedge \overline{x} \leq \overline{y} \\ \text{or } x = \emptyset \end{cases}$$

However, some set operations applied on intervals do not always produce intervals. This happens with the set difference and the union. To keep intervals throughout a sequence of computations, the *hull* operator (denoted by the symbol $\square$) is therefore necessary. The hull of a set is nothing but the smallest interval that contains that set. In particular, the hull of the union of two intervals gives:

$$\square([x] \cup [y]) = \begin{cases} [x] \text{ if } [y] = \emptyset \\ [y] \text{ if } [x] = \emptyset \\ [\min\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}] \text{ otherwise} \end{cases}$$

Given a binary operator $\star$ (like the addition, subtraction, etc.) the set extension of this operator is the operator that takes two sets X and Y and returns

$$X \star Y = \{x \in [x],\ y \in [y],\ x \star y\}.$$

In the case of addition, subtraction and multiplication, which are continuous functions, the set extension of these operators produce intervals when they are applied to intervals. For these operators, the interval arithmetic coincide with the set arithmetic. For two non-empty bounded intervals $[x]$ and $[y]$, these operators have the following explicit formulas:

$$[x] + [y] := [\underline{x} + \underline{y}, \overline{x} + \overline{y}].$$

$$[x] - [y] := [\underline{x} - \overline{y}, \overline{x} - \underline{y}].$$

$$[x] \times [y] := [\min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}].$$

The division is discontinuous in 0 and therefore, the interval arithmetic cannot coincide with the set arithmetic. The division of intervals simply returns the hull of the true set-based division.

$$[x]/[y] = \square\{x \in [x],\ y \in [y],\ x\ /y\}.$$

So, we have an overestimation due to the interval representation.

2.2.1 Interval Vectors

An interval vector $[x] \in \mathbb{IR}^d$ is the Cartesian product of d intervals, that is,

$$[x] = [x_1] \times ... \times [x_d].$$

The interval vectors will be called *boxes*. The components $[x_i]$ of a box are the projections onto the i-th axis (see Figure 2.1). Boxes in our work will usually be 2-dimensional boxes $[x] = [x_1] \times [x_2]$ or 3-dimensional boxes $[x] = [x_1] \times [x_2] \times [x_3]$.

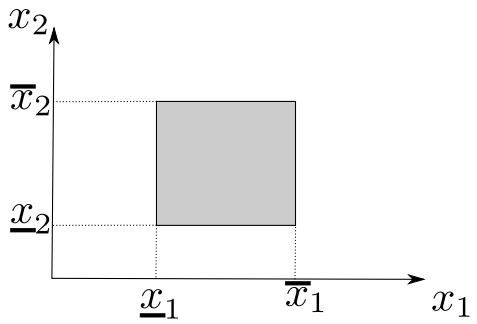


Figure 2.1 – **A *box*** $[x] \in \mathbb{IR}^2$**.**

The width of a box is defined as the largest width of all the components, that is,

$$w([x]) = \max_{1 \le i \le d}\{w([x_i])\}.$$

In the following lines, are listed the main properties and operators for the boxes.

— The middle point of a box $[x]$ is the vector $(mid([x_1]), ..., mid([x_d]))$.

— Given two boxes, $[x] \in \mathbb{IR}^d$ and $[y] \in \mathbb{IR}^d$, the box $[x]$ is a subset of $[y]$ if each of the interval components of the box $[x]$ is a subset of the box $[y]$. Formally, $[x] \subset [y] \Leftrightarrow [x_1] \subset [y_1] \wedge ... \wedge [x_d] \subset [y_d]$.

— Given a vector $x \in \mathbb{R}^d$ and a box $[y] \in \mathbb{IR}^d$, $x \in [y]$ if $x_1 \in [y_1] \wedge ... \wedge x_d \in [y_d]$.

2.3 Inclusion Functions

Let f be a function from $\mathbb{R}^n$ to $\mathbb{R}^m$ (a *real function*) and $[f]$ be a function from $\mathbb{IR}^n$ to $\mathbb{IR}^m$ (an *interval function*).

We say that $[f]$ is an *inclusion function* for f if

$$\forall [x] \in \mathbb{IR}^n, f([x]) \subseteq [f]([x])$$

where $f([x])$ denotes the image of $[x]$ by f, that is

$$f([x]) = \{f(x),\ x \in [x]\}.$$

One of the basic result of interval analysis [Moore 1966] is that an inclusion function can be obtained by replacing each operator of an expression by its interval counterpart.

For instance, the following interval function [Jaulin 2001]:

$$[x] \mapsto [x_1] + [x_2] - [x_1]$$

is an inclusion function for

$$x \mapsto x_1 + x_2 - x_1.$$

An inclusion function can be *pessimistic*, in the sense that for some $[x]$, $[f]([x])$ gives only a coarse approximation of $f([x])$. This effect occurs when at least one variable appears more than once in the expression, like in our previous example. Each occurrence of this variable is then handled independently by the interval operations. To illustrate this, consider our last example. The function $[f]$ actually calculates

$$[x_1] + [x_2] - [x_1] = \{x_1 + x_2 - x_3, \quad x_1 \in [x_1], x_2 \in [x_2], x_3 \in [x_1]\}$$

which, indeed, is an over-approximation of

$$f([x]) = \{x_1 + x_2 - x_1, \quad x_1 \in [x_1], x_2 \in [x_2]\}.$$

Example: with $[x_1] = [x_2] = [0, 1]$, we have $[x_1] + [x_2] - [x_1] = [-1, 2] \supset [0, 1] = f([x])$.

This is called the *dependency effect*. But the main property of $[f]$, the fact that $[f]([x])$ encloses $f([x])$, is always preserved.

Inclusion functions are a fundamental tool as they allow to easily calculate guaranteed enclosures of the image of sets by an arbitrary mapping, would it be nonlinear. The only condition we require is that this mapping is defined by a mathematical expression, formed with the usual operators.

An inclusion function $[f]$ is *thin*, if for any degenerated interval $[x] = x$,

$$[f](x) = f(x).$$

It is *convergent* if for any sequence of intervals $[x]_k$,

$$\lim_{k\to\infty} w([x]_k) = 0 \Longrightarrow \lim_{k\to\infty} w([f]([x]_k)) = 0.$$

Inclusion functions based on interval arithmetic are both thin and convergent.

Another origin of the pessimism in the calculation of $f([x])$ in the case of multivalued functions comes from the simple fact that boxes are used to represent sets. Even if the range of each component of f is tightly enclosed by the corresponding component of the interval vector $[f]([x])$, the image $f([x])$ of a box is usually not a box so the enclosure may actually be very crude. This is called the *wrapping effect*.

2.4 Contractors

Contractors is an important tool in the design of interval algorithms. A contractor is usually associated to a constraint c and basically corresponds to an algorithm that reduces a given box $[x]$ to a new box $[x']$ such that all solutions of c are kept, that is,

$$\forall x \in [x], \quad c(x) \Longrightarrow x \in [x'].$$

However, this definition is often replaced by a more abstract one, where the notion of constraint becomes implicit:

Definition 3 (Contractor [Chabert 2009]) *A contractor is a mapping C from $\mathbb{IR}^n$ to $\mathbb{IR}^n$ such that*

(i)	$\forall [x] \in \mathbb{IR}^n, \mathcal{C}([x]) \subseteq [x]$	*(contraction)*
(ii)	$(x \in [x], \mathcal{C}(\{x\}) = \{x\}) \Rightarrow x \in \mathcal{C}([x])$	*(consistency)*
(iii)	$C(\{x\}) = \emptyset \Leftrightarrow (\exists \varepsilon > 0, \forall [x] \subseteq B(x, \varepsilon), \mathcal{C}([x]) = \emptyset)$	*(continuity)*

where $B(x, \varepsilon)$ is the ball centered on x with radius ε.

We shall however not refer to this definition further and one can simply think about the informal definition given before. The quality of a contractor is its contraction strength and an optimal contractor gives the smallest box, i.e.,

$$\square\{x, x \in [x] \wedge c(x)\}.$$

Consider now the equation

$$c(x) \iff f(x) = 0.$$

The simplest contractor consists in evaluating with interval arithmetic $f([x])$ and checking whether $0 \in f([x])$. If $0 \notin f([x])$, $[x]$ can be contracted to the empty set. Otherwise, it is let unchanged. This contractor can therefore be qualified as *binary* (in the sense that it keeps all or nothing). This test can be easily extended to inequalities.

This contractor can be significantly improved using *backward arithmetic*. The backward arithmetic considers each operator as a *relation*. Consider for instance the addition:

$$z = x + y.$$

Interval arithmetic calculates

$$[z] \leftarrow [x] + [y]$$

which gives the range (or an enclosure of the range) of z for $x \in [x]$ and $y \in [y]$. So $[x]$ and $[y]$ are input intervals while $[z]$ is an output interval which means that the relation is interpreted in a particular or "directed" way. We can consider more generally that we have as inputs three intervals $[x]$, $[y]$ and $[z]$, which are *a priori* estimation of the ranges of x, y and z, and that we can contract each of them from the relation as follows:

$$\begin{aligned}[z] &\leftarrow [z] \cap ([x] + [y]) \\ [x] &\leftarrow [x] \cap ([z] - [y]) \\ [y] &\leftarrow [y] \cap ([z] - [x])\end{aligned}$$

that is, we obtain *a posteriori* estimation of the ranges. The last two operations constitute what we call the *backward arithmetic* because the contraction is made on x and y, in opposition with the *forward arithmetic* which contracts on z.

Forward and backward arithmetics can be used for contracting a box with respect to an arbitrary equation $f(x) = 0$ (or inequality). The principle consists in breaking the expression of the equation into a conjunction of elementary constraints, each involving one operator, such as $z = x + y$, with the cost of introducing intermediate variables, e.g.:

$$z = (x+y)^2 - x \iff \begin{cases} z = w_1 - x \\ w_1 = w_2^2 \\ w_2 = x + y \end{cases}$$

Here, three elementary constraints and intermediate variables w_1, w_2 and w_3 were necessary. Then, the domains of all the variables are contracted by propagating the contractions performed by the forward/backward arithmetic. Propagation is made efficiently by following the structure of the expression. This gives rise to the famous *forward-backward* algorithm or HC4REVISE [Benhamou 1999, Benhamou 2006].

Other techniques exist for building contractors with respect to a set of non-linear constraints. One of the most famous one is the interval Newton operator [Hansen 1978].

In the case of (uncertain) linear equations, well-known contractors are the interval Gauss-Seidel or the Krawczyk iteration [Neumaier 1990].

Furthermore, contractors can also be combined or transformed to produce more sophisticated contractors by applying techniques inspired by CSPs, like propagation, refutation (also called *shaving* or 3B consistency in the context of numerical constraints) [Lhomme 1993, Chabert 2009] or constructive disjunction [Trombettoni 2007, Neveu 2015].

Non-linear constraints can also be rigorously approximated by a linear program, where the feasible region contains the original solution set, using different techniques such as [Lebbah 2004] or [Araya 2012].

2.5 Inner arithmetic

The inner arithmetic is a variant of the backward interval arithmetic that allows to build a sub-box of a box $[x]$ that is inside a given set $\mathcal{S}$ described by inequalities (contrary to the backward arithmetic which gives an enclosure of $[x] \cap \mathcal{S}$). This technique was first introduced in §3 of [Chabert 2010] and used in [Araya 2014] in the context of global optimization. This inner arithmetic can also be used with an

initial point (or initial box) that is *inflated*. The algorithm that inflates a point $\tilde{x}$ is called an *inner inflator*. Formally:

Definition 4 (Inner Inflator) *Let $\mathcal{S}$ be a set.*

An inner inflator w.r.t $\mathcal{S}$ is an operator that takes as inputs:
— *a box $[x]$*
— *a point $\tilde{x} \in [x]$ such that $\tilde{x} \in \mathcal{S}$.*

and returns a box $[\tilde{x}]$ such that
— $\tilde{x} \in [\tilde{x}]$
— $[\tilde{x}] \subseteq [x]$
— $[\tilde{x}] \subseteq \mathcal{S}$,

The inner arithmetic is a set of inflators for elementary functions and operators. To give an intuition of how it works, let us consider first the following elementary function:

$$\mathcal{S} := \{x \mid \sin(x) \in [a, b]\}.$$

Given a point $\tilde{x}$ such that $\sin(\tilde{x}) \in [a, b]$, it is possible to inflate $\tilde{x}$ to an interval $[x]$ inside $\mathcal{S}$ by calculating an integer k such that

$$\tilde{x} \in [\arcsin(a) + 2k\pi, \arcsin(b) + 2k\pi] \quad \text{or} \quad \tilde{x} \in [(2k+1)\pi - \arcsin(b), (2k+1)\pi - \arcsin(a)].$$

The image of the interval on the right side contains $\tilde{x}$ and has its image by the sine function equal to $[a, b]$. Note that there is a unique maximal inflation of a point $\tilde{x}$ inside $\mathcal{S}$, in this example. This maximal inflation is obtained here by finding the right value of k, providing that we also take into account the cases where $a = -1$ and/or $b = 1$ since, in these situations, some intervals (or maybe even all of them) are contiguous and can be merged. Note also that, to obtain numerical guarantee, the floating-point operations has to be rounded in the opposite directions, compared to the classical interval arithmetic: the lower bound has to be rounded *upward* and the upper bound *downward*.

The same can be done for binary operators, like, e.g.,

$$\mathcal{S} := \{x \mid x_1 + x_2 \in [a, b]\}.$$

However, things get now a little bit more complicated since there is not anymore a unique maximal box $[\tilde{x}]$ inside $\mathcal{S}$. Said differently, the maximum inflation becomes a multiple-objective optimization problem. One possible (but very inadequate) choice is to set

$$[\tilde{x}] := [\tilde{x_1}] \times [a - \tilde{x_1}, b - \tilde{x_1}]$$

because, in this case, we trivially get $[x_1] + [x_2] = [a, b]$. Finding a more well-balanced maximal box involves an heuristic choice which requires the knowledge of some bounding *area* where the point has to be inflated. This is the role of the box $[x]$ above.

Finally, the inflation for an arbitrary expression can be made by an induction over the syntactic tree of the constraint expression. The inflation process requires to symbolically inverse each elementary functions

as we have illustrated for the sine function and the addition. This may be called the *forward-backward inflator*.

This inflator has similar properties that the classical forward-backward contractor: the time complexity is in the length of the constraint expression and gives a maximal box $[\tilde{x}]$ (i.e., of maximum size in every dimension) if no variable occurs twice in the expression.

2.6 Sweeping

Now that we have a technique to build an inner box inside $[x]$ that contains a specific point $\tilde{x}$, we can use this service inside a *sweep* loop [Beldiceanu 2001, Chabert 2010]. The sweep loop can be simply viewed as a way to contract a box by "piling up" boxes until some face is entirely covered. This is quickly depicted in Figure 2.2.

More precisely, the sweep loop is a generic technique that allows to build a contractor for a set of constraints, from a set of inflators. The sweep loop requires an inflator for the negation of each constraint. In practice, an inflator for the negation of a constraint is simply obtained by reversing the inequality sign and using the inner arithmetic described previously.

The principle of the sweep algorithm is then to prune a slice of an initial box $[x]$ by forming a union of boxes wich are inside the negation of a constraint, that is, which violate the conjunction. This type of boxes are called *forbidden* boxes (see Figure 2.2).

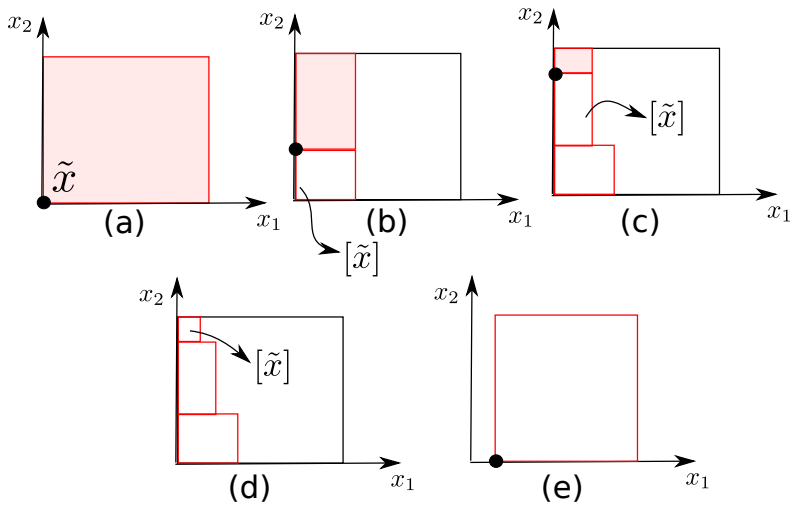


Figure 2.2 – **The sweep loop (2D case)**. The sequence of pictures illustrates a contraction for the lower bound of the variable x_1. At each step, the point $\tilde{x}$ to be inflated is the lower-left corner of the gray box. The inner box $[\tilde{\mathbf{x}}]$ is painted in white. The lower bound of the variable x_1 can be reduced as soon as the projection of the white boxes on x_2 spans the face $[x_2]$, which is the case at step (e).

The sweep loop requires an initial *forbidden* point $\tilde{x}$ (proven to violate at least one of the constraints). This point is successively inflated by all the inflators, and the largest forbidden $[f]$ box is kept.

Once this box is obtained, the next forbidden box is sought in a limited space, called *working area* which is

$$[w] = [\underline{x}_1, \overline{f}_1] \times [\overline{f}_2, \overline{x}_2]$$

This area comes from two reasons. First, the projection on x_1 of the removed slice will inevitably be a subset of $[\underline{x}_1, \overline{f}_1]$ so that any inflation outside $[\underline{x}_1, \overline{f}_1] \times [x_2]$ would eventually be lost. Second, we can take advantage of the first forbidden box to restrict this domain to $[w]$, in order to increase the chance to cover the interval $[x_2]$ entirely with the subsequent forbidden boxes.

So, the working area is updated each time a new forbidden box is obtained. Before the first iteration, the working area $[w]$ is set to the initial box $[x]$.

At each iteration, a forbidden point $\tilde{x} \in [w]$ has to be chosen. Usually $\tilde{x}$ is the lower left corner of $[w]$. The initial forbidden point $\tilde{x}$ is the lower left corner of $[x]$.

Finally, when a forbidden box $[f]$ reaches the limit $\overline{x_2}$, then the following slice can be pruned:

$$[\underline{x}_1, \overline{f}_1] \times [x_2].$$

Of course, it is possible to prune x_2 by changing the order of variables and applying the same procedure.

2.7 Paving strategies

Interval-based techniques allow to represent approximately, but rigorously, an arbitrary subset $\mathcal{S}$ of $\mathbb{R}^d$ described by inequalities.

The set $\mathcal{S}$ is represented by three lists of non-overlapping boxes which form a partition of $\mathbb{R}^d$: the *inner* boxes, the *outer* boxes and the *boundary* boxes. This structure is called a paving [Jaulin 2001]:

Definition 5 (Paving)
A paving of a set $\mathcal{S} \subset \mathbb{R}^d$ is a triplet $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ where $\mathcal{I}$ (for "inside"), $\mathcal{O}$ (for "outside") and $\mathcal{B}$ (for "boundary") are three sets of boxes verifying

$$\mathcal{I} \subset \mathcal{S}, \quad \mathcal{O} \cap \mathcal{S} = \emptyset \quad \textit{and} \quad \mathcal{B} \cup \mathcal{I} \cup \mathcal{O} = \mathbb{R}^d.$$

An example of paving is shown in Figure 2.3. In our context, d will be either 2 or 3.

A paving is always calculated with a precision ε on the boundary, that is, the maximum size of a box that belongs to $\mathcal{B}$.

There are different strategies for calculating a paving, two of them are detailed below. They are based on the operators described earlier in this chapter (contractors, inflators) and on the (middle point) *bisection* operator. This operator takes a box $[x]$ and returns two boxes $L[x]$ and $R[x]$, such that, for some component $j \leq d$,

$$L[x] = [x_1] \times ... \times [\underline{x}_j, \frac{\overline{x}_j + \underline{x}_j}{2}] \times ... \times [x_d]$$

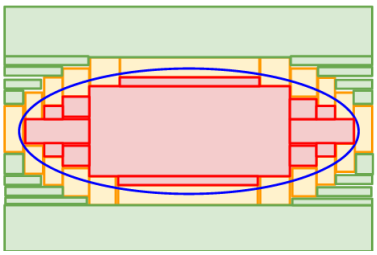


Figure 2.3 – **Paving of an ellipse**. The interior red boxes belongs to $\mathcal{I}$, the unknown yellow boxes in the boundary belongs to $\mathcal{B}$ and the green outer boxes belongs to $\mathcal{O}$.

and

$$R[x] = [x_1] \times ... \times [\frac{\overline{x}_j + \underline{x}_j}{2}, \overline{x}_j] \times ... \times [x_d].$$

The two boxes $L[x]$ and $R[x]$ are called *siblings*. The index j may be chosen according to different policies. One possible and simple policy (which, by the way, ensures convergence of paving algorithms) is to choose the component of $[x]$ of maximal width.

A paving can be efficiently represented as a binary tree, as depicted in Figure 2.4 [Sam-Haroud 1996]. Each node is either a leaf or a bisection node, the latter containing two contiguous nodes resulting from a bisection and representing respectively the left and right subtrees. Leaves represent the final boxes of the paving, as introduced in Definition 5.

The precision ε is used as a criterion for stopping the bisection in a depth-first search and this criterion clearly limits the growth of branches. Another possibility for controlling the growth of the paving is to consider a breadth-first search and to stop bisection when the total surface of the boundary is less than some threshold value.

Note that a tree may be reduced to a single leaf if it is empty of $\mathbb{R}^d$.

The paving is *regular* if each of its boxes can be obtained from $[x]$ by successive middle point bisections. In this case, the *depth* of a box also corresponds to the number of bisections required to obtain that box from the root. A tree is *minimal* if it has no sibling leaves. Any tree can be made minimal by removing all sibling leaves (so that their common parents become leaves). This simply amounts to merging sibling boxes of the paving into single boxes. The IBEX library we have used for our experiments always maintains minimality of pavings on-the-fly.

A paving is calculated by a *paver*. Different implementations of a paver are possible. Two of them are given below, the *bisect & contract* and the *bisect & inflate* strategy.

2.7.1 Bisect & Contract

This first strategy is the classical one [Chabert 2009]. It also corresponds to an extension of the well-known SIVIA algorithm [Jaulin 2001]. It is based on two contractors:

— C with respect to $\mathcal{S}$

— C' with respect to $\neg\mathcal{S}$

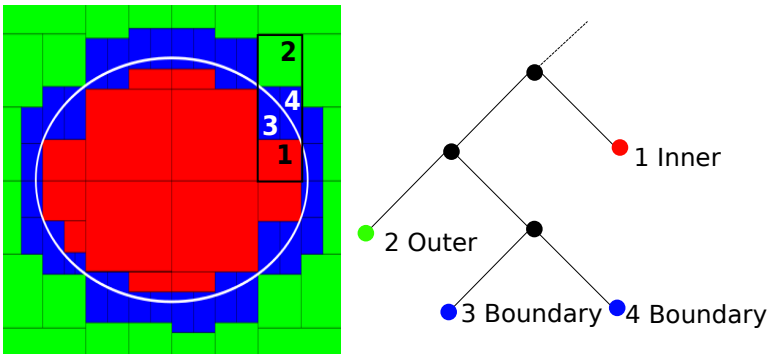


Figure 2.4 – **Paving of an ellipse as a binary tree.** Boxes of $\mathcal{I}$ (resp. $\mathcal{B}$, $\mathcal{O}$) are painted in red (resp. blue, green). On the right side, the corresponding leaves are painted in the same colors (and bisection nodes in black).

The paver is a recursive algorithm that starts with the box $[x] = \mathbb{R}^d$ and alternates three steps:

— *(inner contraction).*
Set $[x^{(1)}] := C([x])$.
If $[x^{(1)}] \neq [x]$ then $\mathcal{I} := \mathcal{I} \cup ([x] \backslash [x^{(1)}])$. Break the remaining part $[x] \backslash [x^{(1)}]$ into boxes and add each subbox to the set of inner boxes.
If $[x^{(1)}] = \emptyset$ then backtrack.

— *(outer contraction).*
Set $[x^{(2)}] := C'([x^{(1)}])$.
If $[x^{(2)}] \neq [x^{(1)}]$, then $\mathcal{O} := \mathcal{O} \cup ([x^{(1)}] \backslash [x^{(2)}])$.
If $[x^{(2)}] = \emptyset$ then backtrack.

— *(bisection).*
If $w([x^{(2)}]) < \varepsilon$ then $\mathcal{B} := \mathcal{B} \cup [x^{(2)}]$.
Otherwise, bisect $[x^{(2)}]$ and perform a recursive call with each box.

2.7.2 Bisect & Inflate

In this thesis, a different implementation of the paver will be used: the *bisect-and-inflate* strategy, which does not actually require a contractor. In fact, we will have at our disposal an outer test, that is an operator $T : \mathbb{IR}^d \to \{\text{true}, \text{false}\}$ such that

$$\forall [x] \in \mathbb{IR}^d, \quad T([x]) = \text{true} \iff [x] \cap \mathcal{S} = \emptyset$$

which proves unsatisfiability in $[x]$; and an inner inflator, that is an operator $In : \mathbb{R}^d \times \mathbb{IR}^d \to \mathbb{IR}^d$ such that

$$\forall [x] \in \mathbb{IR}^d, \ \forall \tilde{x} \in [x] \cap \mathcal{S}, \quad In(\tilde{x}, [x]) \subseteq \mathcal{S}.$$

The paver alternates the following steps:

1. *(outer rejection test).*
   If $T([x]) = \text{true}$ then $\mathcal{O} := \mathcal{O} \cup \{[x]\}$ and backtrack.

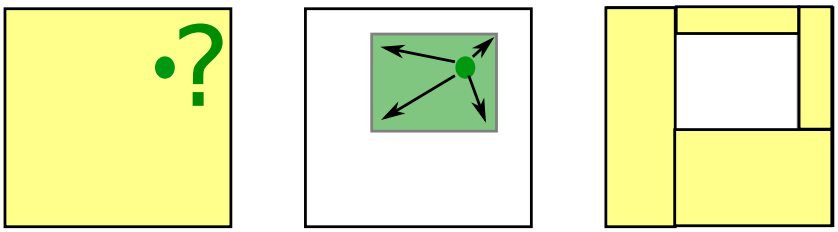
The two others steps are depicted in Figure 2.5. We start by picking randomly a point $\tilde{x} \in [x]$.

If $\tilde{x} \in \mathcal{S}$:

2. *(inner inflation).*
   Set $[\tilde{x}] := In(\tilde{x}, [x])$.
   $\mathcal{I} := \mathcal{I} \cup \{[\tilde{x}]\}$.
   Break the remaining part $[x] \backslash [\tilde{x}]$ into boxes and perform a recursive call with each box.

If $\tilde{x} \notin \mathcal{S}$:

3. *(bisection).* If $[x]$ is small enough, $\mathcal{B} := \mathcal{B} \cup \{[x]\}$.
   Otherwise bisects $[x]$ into two sub-boxes and perform a recursive call with each box.



(a) **(b)** **(c)**

Figure 2.5 – **Inflate & Bisect strategy**. **(a)** Inner test. **(b)** Inner inflation. **(c)** Bisection.

There is another variant: instead of picking randomly a point $\tilde{x} \in [x]$ and checking if it belongs to $\mathcal{S}$, one could rather try to find such inner point by embedding a subsolver (with its own bisection and contraction). However, this subsolver makes somehow a redundant exploration with the one performed by the main search.

2.8 CMA-ES

The CMA-ES algorithm (Covariance Matrix Adaptation Evolution Strategy) [Hansen 2001] has nothing to do with interval techniques. CMA-ES is an evolutionary optimization algorithm.

We have chosen this algorithm for solving the top-level optimization task in our framework (to be detailed in the next chapter). Of course, interval-based optimization algorithms exist [Hansen 1992, Araya 2014]. These algorithms find a global optimizer and bring numerical guarantee. However, they do not accept a black-box function, which is what we have in our framework. Furthermore, they probably run at a higher cost.

We shall give here a few elements about CMA-ES (that can be skipped by the reader).

CMA-ES algorithm is a gradient-free evolutionary algorithm that adjusts the mutation steps with the calculation of a covariance matrix. This process determines the most promising direction, in which the next move is done.

Remember that the concept of *covariance* generalizes the concept of variance to multiple dimensions [Ross 1996]. The covariance between two random variables is a measure of how much these variables change together. A covariance matrix is then composed by the covariances between the i^{th} and j^{th} random variables of a random vector.

The CMA-ES algorithm is an evolutionary algorithm that minimizes a non-linear objective function. To do so, CMA-ES computes an *evolution path*, taken by the population over a number of generations, using an iterative process known as *cumulation*, which successively sums selected mutation steps. A mutation step is taken by stochastic variations of the candidate solutions. These variations are called mutations, and usually are carried out by adding a normally distributed random vector.

Using evolution paths is better than using single mutation steps, because such path contains additional selection information, in contrast to single steps. Indeed, the cumulation process maximizes the probability to reproduce successful evolution paths, whose length can be estimated by the relation between successive mutation steps.

Formally, at each generation g, after the selection of the best current solutions $\{x_k^{(g)}, ..., x_m^{(g)}\}$, the covariance matrix $\mathbf{C}^{(g)}$ of the new distribution is calculated using the current distribution and the previously known evolution path. Modifying the covariance matrix helps to modify the distribution of the mutation steps in the direction of the best known solution. Hence the evolution of the covariance matrix guides the evolution process.

The modifications in the covariance matrix updates the parameters of the normal distribution, used to generate the mutation steps. The mutation steps are guided by normally distributed random vectors. The covariance matrix modifies the variance of the distribution of these vectors. This produces new solutions that can vary more or less with respect to the mean of the W best solutions. If $\langle x \rangle_W^{(g)}$ is the mean of the best W solutions of the generation g, and $r^{(g)}$ is a random vector with distribution $\mathcal{N}(0, \mathbf{C}^{(g)})$, then the new solution $x_k^{(g)}$ is computed as follows

$$x_k^{(g)} = \langle x \rangle_W^{(g)} + r^{(g)}.$$

2.9 Conclusion

In this chapter, we have introduced some interval techniques that we will be used in this thesis. The main algorithms used to handle the overlapping constraint are the contractors and inflators. A contractor is an algorithm that considers a constraint c and reduces a box to a subbox where all solutions of c are kept. In a dual way, an inflator also considers a constraint c and creates a subbox where all the points are solution.

We have also introduced the *sweep* algorithm, an operator which basically transforms a set of inflators into a contractor. This algorithm contracts a box $[x]$ by piling up a set of *forbidden* boxes until one face of $[x]$ is entirely covered. Then a section of that box can be pruned. These forbidden boxes are the result of

the inner inflators, which must then corresponds to the negation of the constraint.

All these algorithms can be used in the context of a paving algorithm. A paving algorithm is a subdivision algorithm that produces a numerical representation of a set. The paving algorithm begins with an initial domain box (usually equal to $\mathbb{R}^d$) and creates recursively three sets of boxes. The first set corresponds to boxes which are inside the set, the second to boxes outside the set and the last to boxes which may cross the boundary. The paving algorithm can have many variants, two of which are considered in particular: the bisect & contract approach and the bisect & inflate approach.

Finally, we have introduced the CMA-ES algorithm, an evolutionary algorithm based on the calculation of the covariance matrix of the mutation steps distribution. This algorithm is used in the top-layer of our packing framework.

In the next chapter, we present how all these components are used to address the generic packing problem.

3

Overall Packing Strategy

3.1 Introduction

The packing algorithm described in this thesis is based on the minimization of an objective function with an evolutionary algorithm.The objective function is an aggregation of so-called *overlapping functions* between all pairs of objects to pack. Finally, an overlapping function is evaluated numerically from the *overlapping region*, a pre-processing of the overlapping constraint. This makes our packing strategy a three-layered algorithm. In this introduction, we rapidly introduce each layer in a top-down way. A more detailed description of each layer is given in the subsequent sections, from bottom to top.

3.1.1 Layer 1: Packing Algorithm

The solution of a packing problem is the given of a configuration for each item to pack (see (1.6)). More precisely, if each item to pack has a configuration $q_i = (o_i, \alpha_i)$, where $o_i \in \mathbb{R}^2$ is the position of the item origin in the plane and $\alpha_i \in [0, 2\pi]$ its rotation angle, a solution for the placement of n items is a $d \times n$-tuple $(q_1, ..., q_n)$.

We model the problem as a continuous optimization problem where the objective function is a combination of the violation cost of each non-overlapping constraint $f_{ij}(q_i, q_j)$. A simple sum can be used so that our objective function to optimize $f(q_1, ..., q_n)$ takes the following form

$$f(q_1, ..., q_n) = \sum_{i,j} f_{ij}(q_i, q_j). \tag{3.1}$$

Once this function is built, the optimization can be externalized to an existing algorithm. However, as we will see, this global cost does not have an analytic expression. In particular, we cannot compute its gradient

which limits our choice to *black-box* algorithms. Furthermore, it should be emphasized that the original continuous packing problem has a very high complexity and is not even in $\mathcal{P}$; indeed, even checking that a non-overlapping constraint is satisfied by a "certificate" (two configurations q_i and q_j) requires us to prove that a non-linear constraint satisfaction problem has no solution, which can be considered as NP-hard (this is indeed already the case for polynomial problems). So we can reasonably say that global optimization algorithm is out of reach. On the other hand, as we will see, using a local optimization algorithm does not mean that we will fail in proving that a solution is safe. It means that we may fail to find such solutions. In other words, we come up with a correct (and rigorous) algorithm but incomplete.

The algorithm we have used in our experiments is CMA-ES (see §2.8).

3.1.2 Layer 2: Overlapping Function

The overlapping function $f_{ij}(q_i, q_j)$ takes as input two configuration vectors q_i and q_j and gives a measure of *how much* Objects i and j overlap. The output must fulfill two properties. First, it has to be 0 iff the two objects are disjoint. Second, it has to decrease when the two objects gets more distant so as to guide the search towards safe configurations.

The overlapping function between two objects can be formally defined as the minimal distance between the current parameters vector q_j and other point q'_j, in the d-dimensional configuration space, such that there is not overlap between the objects i and j when Object j is fixed in q'_j, that is,

Definition 6 (Overlapping Function)

$$f_{ij}(q_i, q_j) := \min\{\|q'_j - q_j\|,\ \neg overlap_{ij}(q_i, q'_j)\}.$$

Note that the distance $\|.\|$ can be either in 2 or 3 dimensions, depending on whether rotation is considered or not. This will be discussed further.

From the definition of the overlapping function, we see that a minimization problem has to be solved every time the function is evaluated. Worse, the function to minimized is subject to a constraint that involves 2 quantified parameters. This amounts to a (d+2)-dimensional bisection process that we shall call the *full-dynamic evaluation scheme*:

Definition 7 (Full-Dynamic Evaluation Scheme)
This scheme evaluates the overlapping function by solving the following optimization problem:

Constants:	$q_i = (o_i, \alpha_i)$ *and* $q_j = (o_j, \alpha_j)$
Variable:	q'_j
Objective:	$\|q'_j - q_j\|$
Constraint:	$\forall p \in \mathbb{R}^2 \quad \neg c_i(R_{-\alpha_i}(p - o_i)) \vee \neg c_j(R_{-\alpha'_j}(p - o'_j))$

We can significantly decrease the complexity with some preprocessing. This is what the *overlapping region* is used for.

3.1.3 Layer 3: Overlapping Region

Running a 5-dimensional embedded minimization process every time the function is evaluated by the top-level optimization algorithm is prohibitive, especially in the case of stochastic algorithms which are known to generate a high number of evaluations.

Our key idea is to alleviate this task by pre-processing the overlapping constraint *in the frame* of the i^{th} object. This makes the number of variables decrease from d+2 to d.

Rigorously, the preprocessing consists in the calculation of the so-called *overlapping region* [Salas 2014] which corresponds to the set of configurations for Object j that make the objects i and j overlap when the configuration of Object i is set to 0 ($o_i = 0$ and $\alpha_i = 0$):

Definition 8 (Overlapping Region)
Given Objects i and j, the overlapping region denoted by $\mathcal{S}_{ij}$ is the following set

$$\mathcal{S}_{ij} := \{q_j \in \mathbb{R}^d \mid overlap_{ij}(0_{\mathbb{R}^d}, q_j)\}. \tag{3.2}$$

This concept is depicted in Figure 3.1.

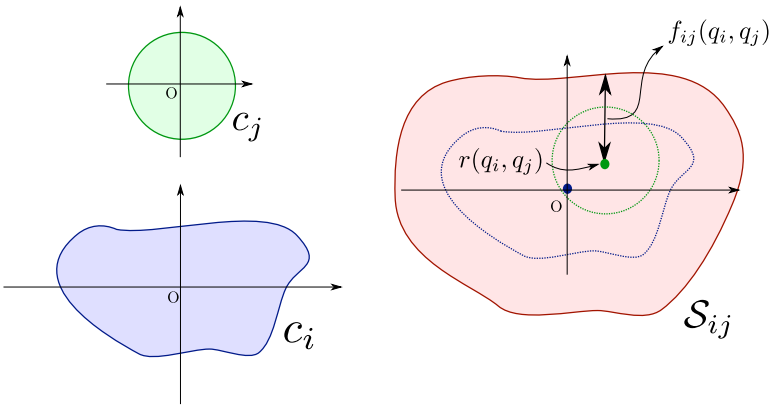


Figure 3.1 – **Overlapping region.** For visibility, the dimension here is 2 (rotation is not considered). **(left)** Objects i (in blue) and j (in green). **(right)** The overlapping region $\mathcal{S}_{ij}$ (in red). Represents all the positions where Object j overlaps Object i, placed at the origin. The distance between the point $r(q_i, q_j)$ and the boundary of $\mathcal{S}_{ij}$ is the value of the overlapping function f_{ij} (Proposition 2).

We shall mention that an exact formula for the overlapping region has been given in [Beldiceanu 2001] in the case where objects are polytopes. The set, in this case, is the convex hull of the points obtained by summing one vertex of the first polytope to one vertex of the second one.

3.2 The Central Proposition

The "central" proposition (Proposition 2 below) makes the connection between the overlapping function and the overlapping region.

It is based itself on a first proposition (Proposition 1) which basically states how the overlapping constraint can be *checked*, given configurations for both Object i and j, using the overlapping region.

Proposition 1
Object j with configuration $q_j = (o_j, \alpha_j)$ overlaps Object i with configuration $q_i = (o_i, \alpha_i)$ iff there exist a vector r in the configuration space (called relative position of Object j with respect Object i) such that

$$r(q_i, q_j) := (R_{-\alpha_i}(o_j - o_i), \alpha_j - \alpha_i) \in \mathcal{S}_{ij}. \tag{3.3}$$

Proof. By definition, $q_j = (o_j, \alpha_j)$ satisfies the overlapping constraint with the object i at position o_i and orientation α_i iff exists $\exists p \in \mathbb{R}^2$ such that $c_i(R_{-\alpha_i}(p - o_i))$ and $c_j(R_{-\alpha_j}(p - o_j))$. This is equivalent to $\exists q \in \mathbb{R}^2$, namely

$$q = R_{-\alpha_i}(p - o_i) \iff p = R_{\alpha_i} q + o_i, \tag{3.4}$$

such that $c_i(q)$ and

$$c_j(R_{-\alpha_j}(R_{\alpha_i} q + o_i - o_j)) \iff c_j(R_{-\alpha_j + \alpha_i}(q + R_{-\alpha_i}(o_i - o_j))). \tag{3.5}$$

This is equivalent to (3.3). ▲

If we assume that checking whether a vector r belongs to $\mathcal{S}_{ij}$ or not can be done in constant time or low complexity, we now have a way to quickly check the satisfiability of the overlapping constraint. That motivates the following proposition:

Proposition 2
Given configurations q_i and q_j for the objects i and j, and $\mathcal{S}_{ij}$ the overlapping region between both objects, we have:

$$f_{ij}(q_i, q_j) = \min\{\|q'' - r(q_i, q_j)\|,\ q'' \notin \mathcal{S}_{ij}\}.$$

Proof. By definition,

$$f_{ij}(q_i, q_j) = \min\{\|q'_j - q_j\|,\ \neg overlap_{ij}(q_i, q'_j)\}.$$

Let us introduce the variable

$$q'' = (o'', \alpha'') := (R_{-\alpha_i}(o'_j - o_i), \alpha'_j - \alpha_i).$$

We have, equivalently,

$$q'_j = (R_{\alpha_i} o'' + o_i, \alpha'' + \alpha_i).$$

In the one hand:

$$\begin{aligned}
&\|q'_j - q_j\|^2 = \|(R_{\alpha_i} o'' + o_i - o_j, \alpha'' + \alpha_i - \alpha_j)\|^2 \\
&= \|R_{\alpha_i}(o'' - R_{-\alpha_i}(o_j - o_i)), \alpha'' + (\alpha_j - \alpha_i))\|^2 \\
&= \|R_{\alpha_i}(o'' - R_{-\alpha_i}(o_j - o_i)\|^2 + |\alpha'' + (\alpha_j - \alpha_i)|^2 \\
&= \|o'' - R_{-\alpha_i}(o_j - o_i)\|^2 + |\alpha'' + (\alpha_j - \alpha_i)|^2 \\
&\quad \text{(because rotation preserves distances)} \\
&= \|q'' - r(q_i, q_j)\|^2.
\end{aligned}$$

On the other hand, by Proposition 1,

$$overlap_{ij}(q_i, q'_j) \iff (R_{-\alpha_i}(o'_j - o_i), \alpha'_j - \alpha_i) \in \mathcal{S}_{ij},$$

that is,

$$overlap_{ij}(q_i, q'_j) \iff q'' \in \mathcal{S}_{ij}.$$

▲

This proposition means that the overlapping function actually maps the current configuration of Objects i and j to the minimal distance between $r(q_i, q_j)$ and the boundary of the overlapping region. This is also depicted in Figure 3.1.

3.3 The Overlapping Region as a Minkowski Sum

We show in this section that a strong connection exists between the concepts of overlapping region and Minkowski sum. This connection may serve as a basis for processing the overlapping region as it will be shown in Section 3.3.4.

This connection is well-known since the early times of geometrical algorithms but, however, only in the case of translated objects. Our contribution in this section mainly lies in the way we have extended this connection to the case of objects that can be both translated and rotated.

3.3.1 The Minkowski Sum

Let us first recall the definition of the Minkowski sum of two sets:

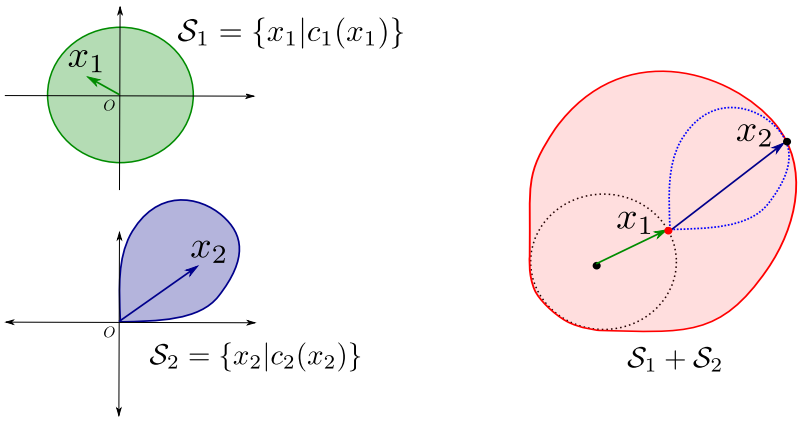


Figure 3.2 – **Minkowski sum.** *Left:* Two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. *Right:* Their Minkowski sum.

Definition 9 (Minkowski sum and difference)
Given two equi-dimensional sets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathbb{R}^d$, the Minkowski sum is

$$\mathcal{S}_1 + \mathcal{S}_2 \triangleq \{x_1 + x_2 \in \mathbb{R}^d : \ x_1 \in \mathcal{S}_1, \ x_2 \in \mathcal{S}_2\}. \tag{3.6}$$

The Minkowski difference is defined accordingly by:

$$\mathcal{S}_1 - \mathcal{S}_2 \triangleq \{x_1 - x_2 \in \mathbb{R}^d : \ x_1 \in \mathcal{S}_1, \ x_2 \in \mathcal{S}_2\}. \tag{3.7}$$

Figure 3.2 shows an example of two sets and their Minkowski sum.

The Minkowski sum has an alternative definition in the case of sets described by constraints:

Proposition 3
If $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) is a d-dimensional set described by a constraint c_1 (resp. c_2), that is, if $x \in \mathcal{S}_1 \iff c_1(x)$ (resp. $x \in \mathcal{S}_2 \iff c_2(x)$) then

$$\mathcal{S}_1 + \mathcal{S}_2 = \{x \in \mathbb{R}^d : \ \exists p \in \mathbb{R}^d, \ c_1(p) \wedge c_2(x - p)\}.$$

Proof.

Substituting p to x_1 in Definition 9, we have

$$\mathcal{S}_1 + \mathcal{S}_2 = \{p + x_2 \in \mathbb{R}^d : \ c_1(p) \wedge c_2(x_2)\}.$$

Said differently,

$$x \in \mathcal{S}_1 + \mathcal{S}_2 \iff \exists p \in \mathbb{R}^d, \ \exists x_2 \in \mathbb{R}^d : \ x = p + x_2 \wedge c_1(p) \wedge c_2(x_2)$$

Since $x = p + x_2 \iff x_2 = x - p$, we have

$$\begin{aligned} x \in \mathcal{S}_1 + \mathcal{S}_2 &\iff \exists p \in \mathbb{R}^d, \ \exists x_2 \in \mathbb{R}^d : \ x = p + x_2 \wedge c_1(p) \wedge c_2(x - p) \\ &\iff \exists p \in \mathbb{R}^d, \ c_1(p) \wedge c_2(x - p). \quad \blacktriangle \end{aligned}$$

Corollary 1
With the same notations as in Proposition 3:

$$\mathcal{S}_1 - \mathcal{S}_2 = \{x \in \mathbb{R}^d : \ \exists p \in \mathbb{R}^d, \ c_1(p) \wedge c_2(p - x)\}.$$

Proof.
This is immediate from

$$x_2 \in (-\mathcal{S}_2) \iff c_2(-x_2). \quad \blacktriangle$$

3.3.2 Overlapping Region as a Minkowski Sum (translation only)

Now, if objects can only be translated, Definition 2 simplifies as follows:

$$overlap_{ij}(o_i, o_j) \iff \exists p \in \mathbb{R}^2, \quad c_i(p - o_i) \wedge c_j(p - o_j). \tag{3.8}$$

Next, fixing the origin of Object i at 0 as in Definition 8, the relation boils down to:

$$o_j \in \mathcal{S}_{ij} \iff \exists p \in \mathbb{R}^2, \quad c_i(p) \wedge c_j(p - o_j). \tag{3.9}$$

We therefore immediately see by Corollary 1 that $\mathcal{S}_{ij}$ is the Minkowski sum of $\mathcal{S}_i$, the set characterized by c_i (the constraint of Object i), and $-\mathcal{S}_j$, the symmetric of the set characterized by c_j (the constraint of Object j). This is illustrated in Figure 3.3.

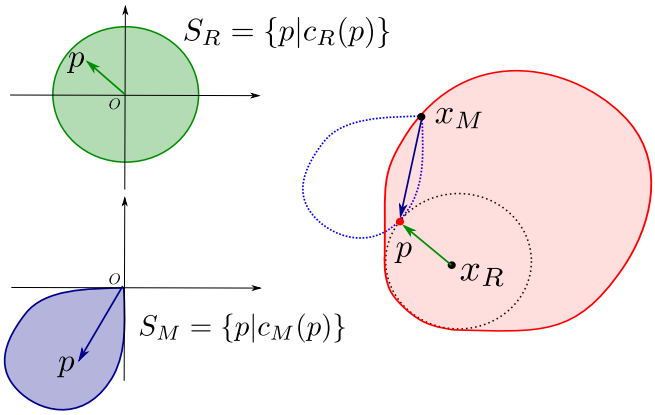


Figure 3.3 – **Overlapping region as a Minkowski sum (translation only).** *Left:* The two objects i and j, which are also two sets $\mathcal{S}_i$ and $\mathcal{S}_j$. *Right:* The non-overlapping region. We recognize the same set as in Figure 3.2; this illustrates that the region coincides with the Minkowski sum of $\mathcal{S}_i$ and $-\mathcal{S}_j$.

3.3.3 Overlapping Region as a Minkowski Sum (translation and rotation)

We show now that reformulating the overlapping region as a Minkowski sum can still be done in the case of rotation, a less trivial result.

To this end, we use the following trick. We embed the object $\mathcal{S}_j \subseteq \mathbb{R}^2$ into $\mathbb{R}^3$ encoding its rotation within the additional dimension:

$$S'_j := \{(v, \beta) : c_j(R_\beta\, v)\} = \{(v, \beta) : R_\beta\, v \in S_j\}. \tag{3.10}$$

Now, the following proposition states that the overlapping region (with rotation allowed) $\mathcal{S}_{ij}$ can be written as a Minkowski difference of two such *augmented* sets (see Figure 3.4).

Proposition 4

$$\mathcal{S}_{ij} = \mathcal{S}_i \times \{0\} - \mathcal{S}'_j. \tag{3.11}$$

Proof 1 *By definition, $(x_j, \alpha_j) \in \mathcal{S}_{ij}$ holds if and only if $\exists p \in \mathbb{R}^2$ such that $c_i(p)$ and $c_j(R_{-\alpha_j}(p - x_j))$. Equivalently, there exist $u_i \in \mathcal{S}_i$ and $u_j \in \mathcal{S}_j$ such that $u_i = p$ and $u_j = R_{-\alpha_j}(u_i - x_j) \iff x_j = u_i - R_{\alpha_j} u_j$. Finally, the vector (x_j, α_j) is proved to be the sum of $(u_i, 0) \in \mathcal{S}_i \times \{0\}$ and $(R_{\alpha_j} u_j, \alpha_j) \in \mathcal{S}'_j$.* ▲

3.3.4 A First Paving Algorithm

Since the overlapping region is equivalent to a Minkowski sum, paving the overlapping region amounts to paving a Minkowski sum. This point of view considerably simplifies the task and we will stick to it here.

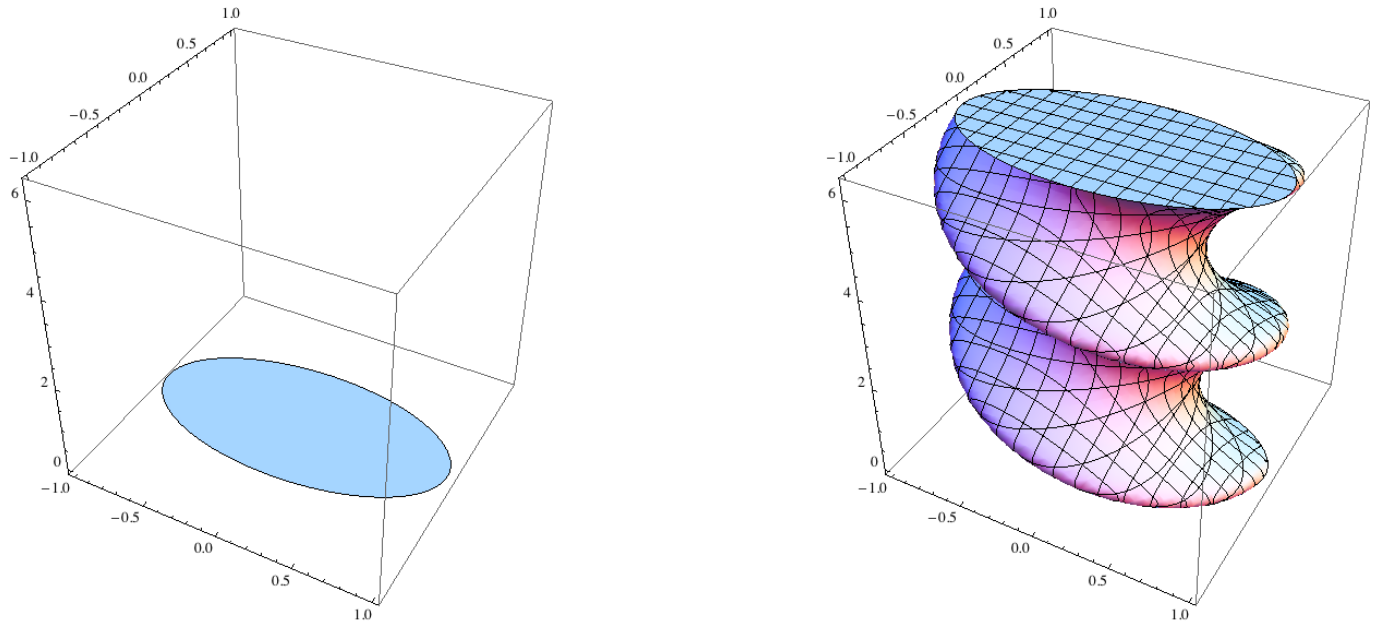


Figure 3.4 – Sets whose Minkowski difference gives the overlapping constraint of two ellipses, when rotation is taken into account. *Left:* the augmented reference ellipse, with rotation coordinate set to 0. *Right*: the augmented moving ellipse $\mathcal{S}'_j$.

In fact, we will propose in Chapters 4 and 5 a different approach that turns out to be more efficient that this one. But this approach is still partly based on the Minkowski sum correspondence so that this section keeps some interest.

So, we assume henceforth that we have to compute a paving of the Minkowski sum

$$\mathcal{S} \triangleq \mathcal{S}_1 + \mathcal{S}_2.$$

The paving structure has been introduced in Section 2.7.

As explained in Section 2.7, calculating a paving requires two algorithms:
— an inner contractor or inflator
— an outer contractor or inflator
We assume here that we already have such algorithms for the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ and we show how we can build algorithms for $\mathcal{S}$. We will see in details how inner inflators and outer tests can be built for objects in the subsequent chapters, depending on the nature of the underlying constraint.

The originality of our approach is essentially in the inner inflator.

3.3.5 Inner Inflator

Before describing how a box $[x]$ can be contracted, let us first address a more simple question: how can we find a subbox of $[x]$ that is inside $\mathcal{S}$?

A possible answer is to look for two boxes $[x]_1$ and $[x]_2$ such that

$$[x]_1 \subseteq \mathcal{S}_1, \quad [x]_2 \subseteq \mathcal{S}_2 \ \wedge \ ([x]_1 + [x]_2) \cap [x] \neq \emptyset \tag{3.12}$$

because, in this case,

$$([x]_1 + [x]_2) \subseteq [x] \cap \mathcal{S}.$$

To find such boxes, one can calculate in parallel two pavings, one of $\mathcal{S}_1$ and one of $\mathcal{S}_2$, and stop the process as soon as two boxes satisfying (3.12) are found. By combining boxes of the first paving with boxes of the second one, this approach amounts to running a branch & bound in a $(2 \times d)$-dimensional space. Note that this branch & bound is a sub-solver embedded in the main one, the one for the x variable. Our goal is to reduce the sub-solver to d dimension only, which is, by the way, the incompressible price to pay for handling d existentially-quantified parameters. Indeed, remember that the overlapping constraint involves a parameter p from the very beginning (see Definition 2), and this parameter may have up to $d = 3$ components in the case of augmented objects.

To reduce the complexity, we use the same idea as above, but this time based on Proposition 3. The procedure explained below is also depicted in Figure 3.5.

To build an inner box in $[x]$, let us first assume that we have picked some point $\tilde{x}$ inside $[x]$ (this point is, in fact, automatically yielded by the sweep loop, as it has been explained in Figure 2.2). Then we look for another point $\tilde{p}$ such that

$$c_1(\tilde{p}) \wedge c_2(\tilde{x} - \tilde{p}). \tag{3.13}$$

Finding this point is the task of the subsolver. Note that the subsolver is now dedicated to a different task and works differently. Since only one point is sought at the intersection of both sets, at each node of the search, we simply check membership to $\mathcal{S}_1$ and $\mathcal{S}_2$ with a point $\tilde{p}$ picked randomly in the current domain. If both are satisfied, the search is interrupted and $\tilde{p}$ is returned. The depth of the search is also controlled by a precision on the domain width, which ensures that the subsolver terminates in bounded time. In case of normal termination, no $\tilde{p}$ hence no inner box has been found. This kind of algorithm is classical and very easy to achieve with interval techniques, especially in the case of sets described by inequalities.

Once $\tilde{p}$ is found, it is *inflated* to a subbox $[x]_1$ of $(\tilde{p} + [x] - \tilde{x})$ that is inside $\mathcal{S}_1$, using the inner inflator with respect to $\mathcal{S}_1$. The point $(\tilde{x} - \tilde{p})$ is also inflated to a sub-box $[x]_2$ of $([x] - \tilde{p})$ that is inside $\mathcal{S}_2$. However, if $\tilde{x}$ turns out to be outside $\mathcal{S}$, the last inflation cannot succeed and the process is interrupted in this case. Otherwise, the resulting box satisfies $([x]_1 + [x]_2) \subseteq \mathcal{S}$ and $([x]_1 + [x]_2) \cap [x] \neq \emptyset$.

Note that $([x]_1 + [x]_2) \cap [x] \neq \emptyset$ is just a consequence of $\tilde{x} \in [x]$. So the initial boxes used for both inflations are somehow arbitrary, but fixing them as we did is a heuristic that tends to maximize the surface of the final inner box $([x]_1 + [x]_2) \cap [x]$.

3.3.6 Outer Rejection Test

Rejecting a box $[x]$ means proving $[x] \not\subseteq \mathcal{S}$, that is

$$\forall x \in [x] \quad \forall p \in \mathbb{R}^d, \ \neg(c_1(p) \wedge c_2(x - p)). \tag{3.14}$$

This assertion can be checked by running the same subsolver we used for the inner inflator, except that the point $\tilde{x}$ is replaced by the current box $[x]$. If the subsolver finds no solution, the previous assertion is proven.

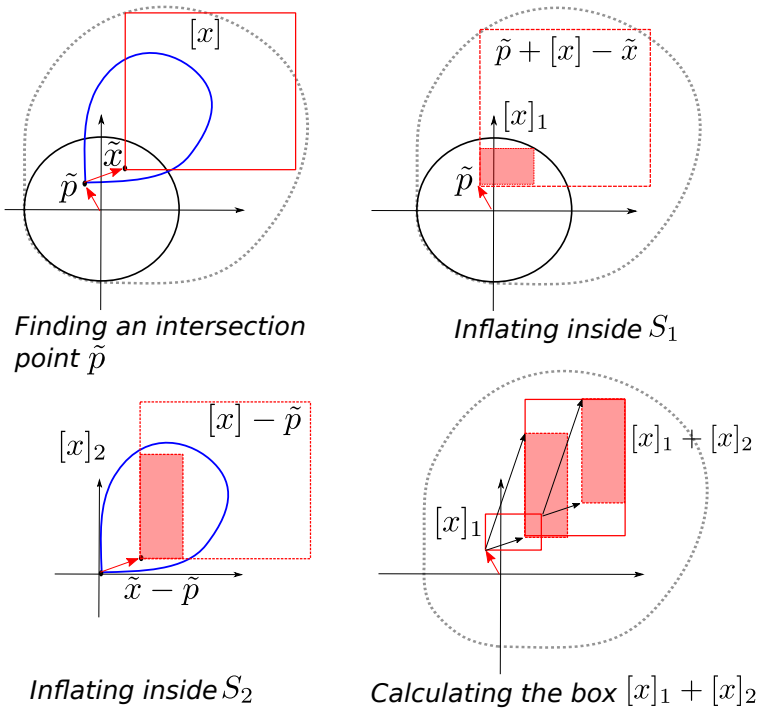


Figure 3.5 – **Steps of the inner Inflator.**

Note that only the coordinates of p are bisected, so the subsolver actually proves a stronger assertion if the contraction with respect to c_2 is not optimal (the actual assertion depends on the consistency level enforced by the contraction with c_2).

Note also that the precision used in the subsolver is dynamically set to the width of $[x]$ in order to have a somewhat uniform time spent by the subsolver throughout the global search (on x). This dynamic precision also ensures that the outer contractor is *convergent*, that is, it rejects any sufficiently small boxes outside $\mathcal{S}$.

One may be surprised by the simplicity of this rejection test and expect a more elaborated contractor for the outer region, inspired by what we did for the inner region. But the situation could be interpreted in the other way around. Since the overlapping constraint is in two dimensions only, an inner satisfiability test would probably fits, as long as it is fast and convergent. However, such a test amounts to proving for $[x] \subseteq \mathcal{S}_1 + \mathcal{S}_2$ the following assertion

$$\forall x \in [x] \quad \exists p \in \mathbb{R}^d,\ (c_1(p) \wedge c_2(x - p))$$

and, contrary to (3.14), the $\forall$ and $\exists$ quantifiers are involved, which means that the problem is much harder. So, the inner contractor can be seen here as a way to make up for the lack of inner test.

Our previous argument is only based on running time. It is clear that an outer contractor could also lead to a more compact paving, but the size of the paving is anyway conditioned by the representation of the boundary so that a drastic gain on this aspect is not really expectable.

3.3.7 Object Preprocessing

The Minkowski sum of two sets can also be computed with an (apparently) very simple approach.

The idea is simply to calculate pavings for the two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ (that is, to pre-process the objects i and j), and to perform the addition of the pavings.

Indeed, if we denote by $(\mathcal{I}_1, \mathcal{B}_1, \mathcal{O}_1)$ and $(\mathcal{I}_2, \mathcal{B}_2, \mathcal{O}_2)$ the two pavings (see §2.7 for the notations), it is clear that we have the following formula:

$$\mathcal{I}_1 + \mathcal{I}_2 \subseteq \mathcal{S} \subseteq (\mathcal{I}_1 \cup \mathcal{B}_1) + (\mathcal{I}_2 \cup \mathcal{B}_2)$$

which means that the set $\mathcal{S}$ can be obtain by calculating each of these "bounds" (in the sense of the inclusion order). It is trivial, by the way, to perform the addition of two sets of boxes. One just has to take the union of the sum of every pairs of boxes formed with one box from each set. However, this union is a set of overlapping boxes. So a treatment is necessary to get a paving. It turns out that the best way we come up with for producing a paving from that formula is actually to run a bisect-and-contract scheme for $\mathcal{S}$ and to use an inner contractor:

$$[x] \mapsto [x] \backslash \Big([x] \cap (\mathcal{I}_1 + \mathcal{I}_2) \Big)$$

and the outer contractor:

$$[x] \mapsto [x] \cap (\mathcal{I}_1 \cup \mathcal{B}_1) + (\mathcal{I}_2 \cup \mathcal{B}_2).$$

This somehow leads to a scheme with three embedded loops, one for bisecting boxes $[x]$, one for enumerating boxes over $\mathcal{S}_1$ and one for enumerating boxes over $\mathcal{S}_2$. This is a higher complexity than the inflate-and-bisect strategy proposed above, which basically amounts only to two embedded loops (one corresponds to the embedded sub-solver that looks for the point $\tilde{p}$).

Furthermore, using pavings for objects introduce another level of approximation, since pavings describe objects up to some precision.

It is interesting to note that both approaches naturally rule out the *fake boundary* issue (see §4.3 for more details) as the sum of two boundary boxes that lies inside the sum of two inner boxes is classified as inner.

3.4 The Overlapping Function

3.4.1 Some Important Properties

Let us recall the definition:

$$f_{ij}(q_i, q_j) := \min\{\|q_j' - q_j\|,\ \neg overlap_{ij}(q_i, q_j')\}.$$

As already said, the norm $\|.\|$ mixes distances and angles which is somehow both a drawback and advantage. The drawback is that the calculation of the violation cost is arbitrary in essence since one has to

basically set the weighting of angle unities (*radians*) with respect to the distance unity (like *meters*), in the definition of the norm.

The advantage is that by tuning this weighting, we have a direct control on the black-box optimization behavior. The more this weighting, the more the optimization tries to obtain a valid packing by turning objects around their position; and the less this weighting, the more it proceeds by translating them.

This definition also introduces a distinction between the two shapes. Object i is called the *reference* object and Object j the *moving* object. Without rotation, the roles are symmetric in the sense that:

$$f_{ij}(q_i, q_j) = f_{ji}(q_j, q_i).$$

However, the previous equality does not hold with rotation. Henceforth, we will always assume Object i (resp. j) to be the reference (resp. moving) one.

It is important to notice that our definition of the overlapping function does not coincide with the *overlapping surface*, as Figure 3.6 shows. Furthermore, taking the overlapping surface as a measure of overlapping would not be appropriate since the surface can reach a local minimum in a situation where the two objects still overlap (see Figure 3.6). In contrast, our overlapping function only reaches a minimum at 0, that is, when the objects are disjoint. This is an important difference as this function is eventually called by a local search that can be trapped in local minima.

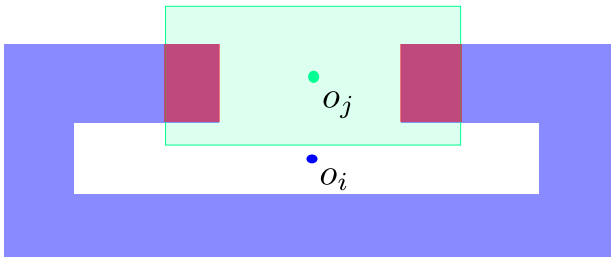


Figure 3.6 – **Overlapping surface versus overlapping function.** The first object is the rectangle frame in blue, the second the rectangle in green. In this situation, the overlapping surface, in red, reaches a local minimum while the objects still overlap.

3.4.2 Relation with Penetration Depth

We have defined the overlapping function as the distance between the current configuration for Object j, and its closest configuration that satisfies the non-overlapping constraint. This way, the function f_{ij} can be seen as a *distance to satisfaction* for the Object j, the configuration of Object i being considered as fixed. This function exactly matches the concept of *penetration depth* in the realm of computational geometry [Cameron 1986, Dobkin 1993, Kim 2002, Kim 2004].

3.4.3 Distance to Boundary

Let us now explain how the overlapping function is evaluated once a paving has been calculated for the overlapping region. This will result in the *Semi-Static Evaluation Scheme*.

We assume here that a paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ exists for the overlapping region.

We make use of Proposition 2 and set $r := r(q_i, q_j)$. Then, we find the closest point to r which is not inside the overlapping region. However, because the boundary of the region is uncertain, we can actually only obtain an enclosure of this minimal distance:

$$\min_{q'' \in \cup \mathcal{B}} \|q'' - r\| \leq f_{ij}(q_i, q_j) \leq \min_{q'' \in \cup \mathcal{O}} \|q'' - r\|.$$

Clearly, the accuracy of the enclosure is related to the precision ε the paving of the overlapping region has been generated with.

In practice, to calculate a bound, say the upper one, we consider each box $[q'']$ of the set $\mathcal{O}$ generated by the paver, and minimize the distance over that box. That is, we calculate:

$$\text{mindist}(r, [q'']) := \min_{q'' \in [q'']} \|q'' - r\|. \tag{3.15}$$

Then, we have:

$$\min_{q'' \in \cup \mathcal{O}} \|q'' - r\| = \min_{[q''] \in \mathcal{O}} \text{mindist}(r, [q'']). \tag{3.16}$$

The same holds for the lower bound.

Formula (3.15) is nothing but the distance between a box and a point. This distance can be easily obtained in constant time (whatever is the size of the box), either using interval arithmetics or by a direct geometric argument.

Formula (3.16) can however be very time-consuming if the boxes in $\mathcal{O}$ are stored in a flat unsorted list. One has to systematically scan the whole list to get the minimal distance. This linear complexity can be easily transformed to a logarithmic complexity using a tree-structure representation, as it has been depicted in Figure 2.4. The structure is then explored using standard global optimization techniques. In particular, pending nodes are stored in a heap, the criterion associated to a node being precisely its distance to r. Nodes with a distance exceeding the current upper bound for the minimum can then be discarded (just like the *bounding* step in a branch & bound).

In summary:

Definition 10 (Semi-Static Evaluation Scheme)
Set

$$r := r(q_i, q_j)$$

and then solve the following optimization problem (in $\mathbb{IR}^d$):

Variable:	$[q'']$
Objective:	$\text{mindist}(r, [q''])$
Constraint:	$[q''] \in \mathcal{O}$

The embedded optimization problem is now far more easy to solve than the one in the full-dynamic evaluation scheme.

3.4.4 Other Alternatives

We shall briefly evoke here two other evaluation schemes that may appear in one's mind and explain why they have not been selected.

The first option is the *Full Static* one. Instead of pre-processing the overlapping region between two objects and then calculating on-line the distance to the boundary for input vectors r, one could imagine that a paving of this distance function itself (i.e., the overlapping function) could be computed. This would result in a piecewise function, as depicted in Figure 3.7.

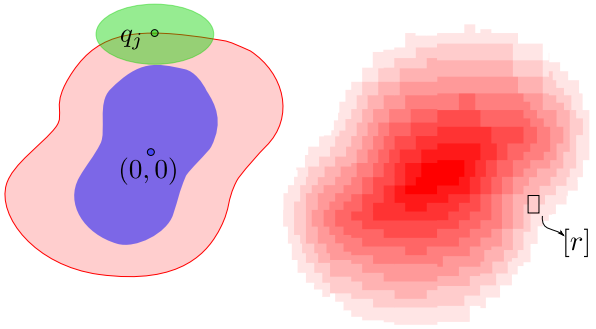


Figure 3.7 – **A paving of the overlapping function.** The color nuances on the right picture show the different isolines of the overlapping function, the darker the higher. White color means zero.

This approach is valid, of course, but it has been ruled out for the following reason. If one wants to keep the size of the paving reasonable, the most consistent strategy is to stop bisecting a box $[r]$ when the relative uncertainty on the minimal distance is less than, say, 1%. Rigorously, if we denote

$$l([r]) := \min_{r\in[r]} \min_{q''\notin\mathcal{S}_{ij}} \|r-q''\| \quad \text{and} \quad u([r]) := \max_{r\in[r]} \min_{q''\notin\mathcal{S}_{ij}} \|r-q''\|.$$

Then the box is not bisected if $u - l \leq \frac{l}{100}$. This is, by the way, what we implemented for producing Figure 3.7.

The problem is that local optimization softwares (like CMAES) work by locally calculating a quadratic approximation of the function. The quadratic model is obtained by sampling the function at different points and we have no a priori information on the size of the step. In particular, there is no evidence that the step would be large if the value of the function is far from zero. If the local optimization algorithm samples several point inside the same box $[r]$ of the paving, the function is viewed as completely flat, resulting to a spurious minimum. Except if we dig into the details of the software, there is no known relation between the size of the step and the value of the criterion. Assuming such relation would clearly affects the simplicity and generality of our approach.

At the extreme opposite, another alternative is the *Semi-Dynamic* one. The principle of this approach is to calculate the paving of $\mathcal{S}_{ij}$ *while* minimizing the distance between its complementary and r. We have implemented it and realized that the time spent for running contractors and inflators inside the minimization process was prohibitive. Furthermore, this approach raises some difficulties as the set we calculate the distance to is initially unknown. For instance, if we give priority in the search to the box which is the

closest to r (as done in the previous subsection), then the search cannot find any box of $\mathcal{O}$, as there always exists a box closer to r that crosses the boundary. This means that the upper bound u is never updated, leading to an infinite loop.

3.5 Conclusion

In this chapter we have presented our framework for solving the packing problem, which is split in three layers: the *packing algorithm*, the *overlapping function* and the *overlapping region*.

The packing algorithm solves the packing problem by minimizing a violation cost. It performs a local search which means that the configuration vectors for each object is always fixed. This is a fundamental point as the subsequent techniques cannot deal with uncertain domain for configurations.

The violation cost is just the addition of overlapping costs for all pairs of objects to pack. The value of this violation cost with respect to the configurations is called *overlapping function* and is defined as the minimal distance (in the configuration space of one object) required to separate the two objects. To obtain quickly this distance, we pre-calculate all the configurations where the two objects overlap, the position of one being fixed as the origin. This set of points is called *overlapping region*. A numerical representation of this region is computed using the paving algorithm described in Chapter 2.

The overlapping function and the overlapping region are connected by Proposition 1 which considers the relative configuration $r(q_i, q_j)$ of the object j with respect to the object i. This point belongs to the overlapping region if and only if both objects overlap. Consequently, the overlapping function can be computed as the minimal distance between r and a point outside the inner part of the overlapping region. This definition of the overlapping function is close to the concept of *penetration depth*.

In this chapter, we have also explored the well-known relation between the overlapping region and the Minkowski sum. We have proposed a way for extend this relation in the case of rotated objects, by considering so-called *augmented objects* where the rotation angle is encoded in an additional geometrical dimension.

We have finally proposed a first paving algorithm, derived from the Minkowsk sum equivalence, and detailed the three ingredients: inner inflation, outer rejection test and bisection. The originality of the algorithm lies in the inner inflation algorithm for the overlapping constraint, which simply exploits the fact that the sum of two boxes, each included in a different set, is included in the Minkowsk sum of that sets.

What is left then is the ability to inflate or contract with respect to objects. The solution depends on the representation of the objects. In the following chapters, we will present the overlapping region computation when the objects are either described by inequalities (Chapter 4) or parametric curves (Chapter 5). In fact, a more efficient alternative to the Minkowski sum approach will also be proposed.

4

Objects Described by Non-Linear Inequalities

4.1 Introduction

The goal of this chapter is to instantiate our general packing strategy to the case of objects described by non-linear inequalities.

The basic case is an object of the following form:

$$c(p) \iff f(p) \leq 0.$$

There is no assumption on the function f involved, except that it is a well-identified mathematical expression based on usual operators ($+$, $\times$, $\sqrt{.}$, $\exp$, etc.).

The symbol f should not be confused with the overlapping function of Chapter 3, function to which will we not have to refer anymore in this chapter.

This representation choice encompasses basic geometric shapes like half-planes or ellipses. E.g., a circle of radius 1 and which origin is the center is represented by an inequality $f(p) \leq 0$ with

$$f : p \mapsto \|p\| - 1.$$

We also accept the logical and/or operators in the shape description, which means that these geometric primitives can be combined to define more complex shapes like a polygon, a half-circle or a *horseshoe*, like in Figure 1.3.

The general form of objects that can be handled in our framework is:

$$\begin{aligned} c(p) \iff \quad & (f_{11}(p) \leq 0 \wedge \ldots \wedge f_{1n_1}(p) \leq 0) \ \vee \ldots \\ & \ldots \vee \ (f_{m1}(p) \leq 0 \wedge \ldots \wedge f_{mn_m}(p) \leq 0). \end{aligned}$$

Note that non-linear inequalities give also freedom to represent a lot of fancy shapes but real-world objects are rather obtained by combining basic primitives.

For the sake of simplicity, the results below will be presented assuming one inequality but the reader should keep in mind that these results can always be easily extended to the more general case of disjunctions of conjunctions of inequalities.

Remember that two algorithms need to be specialized: the inner inflator and the outer test. The inner inflation is more difficult to design and leads to a greater computational burden. So the "quality" of this inflator plays a key role in the paving algorithm and has a stronger impact on the efficiency.

We first follow the "Minkowski sum" approach and then describe another inflation technique that substantially improves the performances.

4.1.1 Inflator for the Augmented Object

In Section 3.3.1, we have built an inner inflator for the overlapping region $\mathcal{S}_{ij}$ in a generic way. This inflator requires the ability to inflate a vector $\tilde{p}$ inside the objects $\mathcal{S}_i$ and $\mathcal{S}_j$, or the augmented objects $\mathcal{S}'_j$ and $\mathcal{S}'_j$ in case of rotation.

As objects are described by inequalities, we can directly inflate $\tilde{p}$ inside an object using generic inflation technique for inequalities based on inner arithmetic, as introduced in Section 2.5.

This perfectly works with translation. In the case of rotation, however, this is much less satisfactory. The point is that the inner arithmetic is very sensitive to the number of times a variable appears in the equation, and the angle α has inevitably many occurrences due the rotation matrix R_α. Worse, the inflator inflates in all the dimensions of q simultaneously with a single mathematical expression and the pessimism of the angle combines with that of all the other variables.

To fix ideas, assume an object is described by:

$$c(p) \iff p_x^2 + p_y^2 + p_x p_y - 1 \leq 0.$$

Then, the augmented object is:

$$c'(\alpha, p) \iff c(R_\alpha\, p) \iff c\left(\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \times p\right)$$

$$\iff \begin{array}{l} (\cos(\alpha)p_x - \sin(\alpha)p_y)^2 + (\sin(\alpha)p_x + \cos(\alpha)p_y)^2 \\ +(\cos(\alpha)p_x - \sin(\alpha)p_y) \times (\sin(\alpha)p_x + \cos(\alpha)p_y) - 1 \leq 0 \end{array}.$$

It is true that part of the pessimism can be counter-balanced using a DAG structure for the expression. But one should be convinced that interval arithmetic will anyway hardly work unless some domains are very thin. So, although valid in theory, the Minkowski sum approach is not efficient in practice when the dimension is 3 (i.e., with rotation handled).

Some performances with this approach are reported in the experiments section; they show that the time required to calculate 3D pavings with this inflator is indeed prohibitive.

4.2 A Two-Steps Inflation

The inflator we propose now builds directly a box $[q_j]$ inside the overlapping region $\mathcal{S}_{ij}$, from a single configuration vector $\tilde{q_j}$.

Our new inflator does not suffer from the multiple symbol occurrences introduced by rotation matrices. It splits the inflation in two steps: it first inflates with respect to o and then with respect to α. We describe both steps and then show how the two inflations can be combined.

4.2.1 Inflation w.r.t. Translation

The inflation w.r.t. o is actually very similar to the inflator proposed in Section 3.3.5. It is however not necessary to think in terms of Minkowski sum to explain this part of the inflation and a more direct presentation is preferred here.

The steps are illustrated in Figure 4.1. Given the initial configuration $(\tilde{o_j}, \tilde{\alpha_j})$ we first look for a vector $\tilde{p}$ that belongs to both objects, the configurations of Objects i and j being respectively 0 and $\tilde{q_j}$, that is, $\tilde{p}$ satisfies

$$c_i(\tilde{p}) \wedge c_j(R_{-\tilde{\alpha_j}}(\tilde{p} - \tilde{o_j})).$$

Let us then define $u := \tilde{p} - \tilde{o_j}$ and assume $\tilde{p}$ is inflated to a box $[p]$ inside the i^{th} object (see Listing 4.1). The box $[p]$ can be translated to $\tilde{o_j}$ in the sense that if o_j moves inside $[o]_j := [p] - u$ then $o_j + u$ is both inside $[p]$ (hence inside Object i) and inside Object j with configuration $(o_j, \tilde{\alpha_j})$.

Again, this technique is very close to calculating a Minkowski sum. Note, however, that there is no inflation this time with respect to the second object (Object j). The reason is that this additional inflation would prevent it from being combined with the inflation for the angle, presented in the next section. Clearly, if no rotation occurs, one should allow this additional inflation. That is why, in Listing 4.3, a distinction will be made depending whether rotation occurs or not.

Listing 4.1 – Inflation inside an object

```
/**
 * \parameter c:      The object
 * \parameter q̃:      Configuration of the object (position and
 *                    rotation angle)
 * \parameter p̃:      Point to inflate
 * \parameter [p]:    Domain inside which inflation should be done
 *
 * \return             The inflated box
 */
Box inflat_object(Object c, Point q̃, Point p̃, Box [p]) {

        // Inflator for the constraint c(R_{-α}(o - p)).
        // Variables are o ∈ ℝ², α ∈ ℝ and p ∈ ℝ² (in this order).
        // All except p will be fixed.
        Inflator inf(c)

        // First argument of inf: box inside which inflation occurs
```

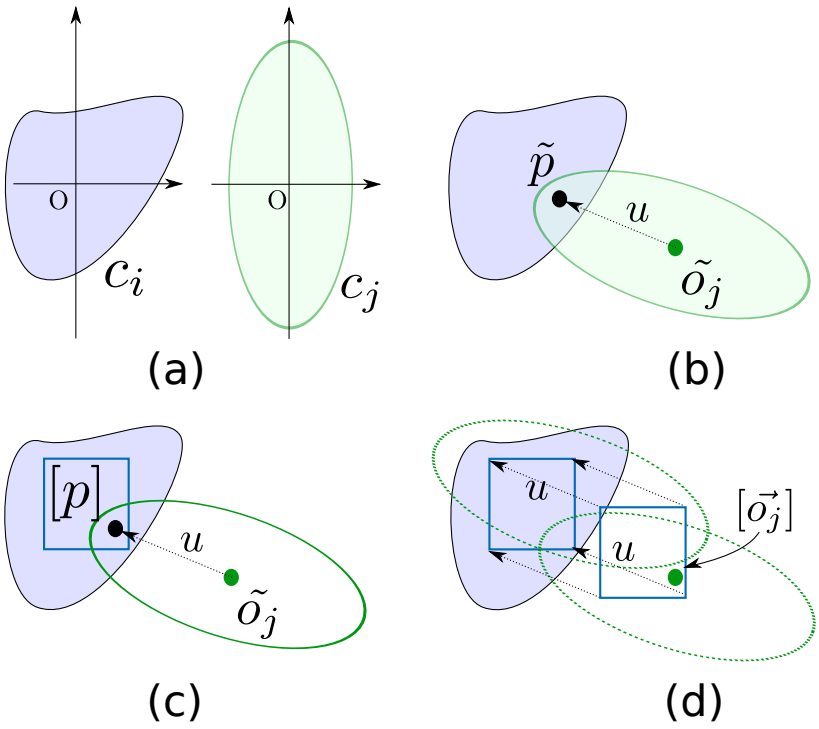


Figure 4.1 – **Inflation w.r.t. translation.** **(a)** The shapes of Objects i (in blue) and j (in green). **(b)** A point $\tilde{p}$ at the intercection of the two objects when the configuration of Object j is $(o_j, \tilde{\alpha_j})$. **(c)** Inflation of $\tilde{p}$ inside Object i. **(d)** Inflation of o_j.

```
        // Second argument: point to inflate
        (q̃, [p]′) = inf.inflate(q̃ × [p], q̃ × p̃)

        return [p]′
}
```

4.2.2 Inflation w.r.t. Rotation

The inflation for the angle is depicted in Figure 4.2 and works as follows.

Given the initial configuration $(\tilde{o_j}, \tilde{\alpha_j})$ we look for the two angles $\underline{\alpha}$ and $\overline{\alpha}$ that force vector $\tilde{p}$ (the same as before) to meet the boundary of Object j (this step is detailed in the next section). Then, it is clear that for every angle $\alpha_j \in [\underline{\alpha}, \overline{\alpha}]$ there is an overlapping.

However, some caution is required.

— First, there are generally more than 2 candidate angles (see Figure 4.2.d), especially if Object j is non-convex. The angle values that form the narrowest interval around $\tilde{\alpha_j}$ are kept.

— Second, there may be no angle at all. This means that Object j can make a complete turn while containing $\tilde{p}$ (or, in other words, $\tilde{p}$ is in the inscribed circle of the object).

— Third, we have to take into account the 2π-periodicity. Indeed, the search space for angles is a bounded domain like $[-\pi, \pi]$ or $[0, 2\pi]$ and this convention has to be decided once for all at the very

top level (when the packing problem is modeled). This has the unpleasant consequence to make the inflated domain disconnected, e.g., if $\tilde{\alpha_j} = \frac{\pi}{2}$, $\underline{\alpha} = 0$ and $\overline{\alpha} = \frac{3\pi}{2}$ then the inflated interval $[0, \frac{3\pi}{2}]$ becomes $[-\pi, -\frac{\pi}{2}] \cup [0, \pi]$. So, in this case, we only keep the subinterval containing $\tilde{\alpha_j}$, that is, $[-\pi, -\frac{\pi}{2}]$.

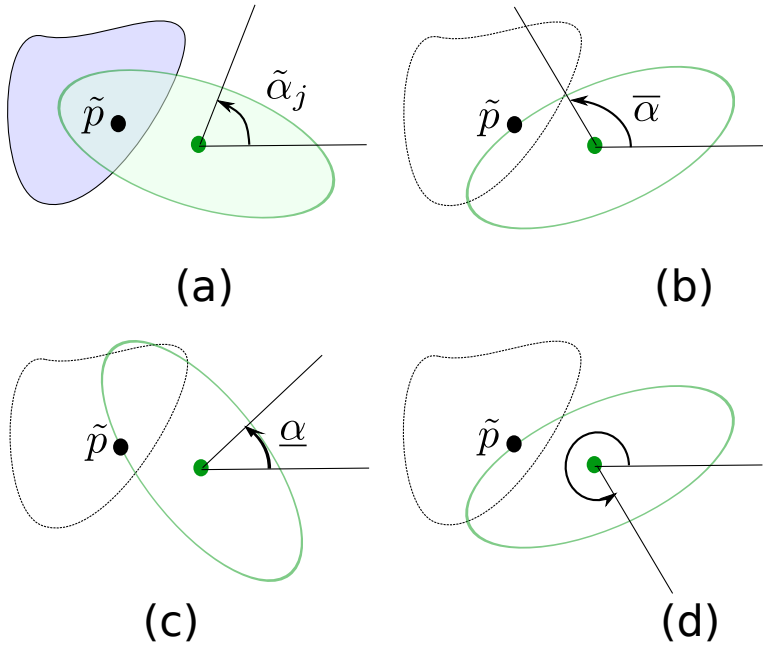


Figure 4.2 – **Angle inflation.** **(a)** Point $\tilde{p}$ obtained with initial configuration (translation step). **(b)** The smallest angle greater than $\tilde{\alpha_j}$ that makes $\tilde{p}$ meet the boundary of object j. **(c)** The greatest angle lower than $\tilde{\alpha_j}$ with the same property. **(d)** Another angle with the same property (there are 4 like this) that is discarded.

4.2.3 Shape Boundaries

Let us explain now how the angles $\underline{\alpha}$ and $\overline{\alpha}$ are calculated. First, we have to generate a constraint c'_j for the boundary of Object j. In the simplest case of a single inequality,

$$c_j(p) \iff f(p) \leq 0,$$

the boundary of the object is described by the equality $f(p) = 0$.

In the case of a conjunction of inequalities,

$$c_j(p) \iff (f_1(p) \leq 0 \wedge \ldots \wedge f_n(p) \leq 0).$$

The boundary is described by a disjunction of n systems:

$$\begin{array}{l} c'_j(p) \iff (f_1(p) = 0 \wedge f_2(p) \leq 0 \ldots \wedge f_n(p) \leq 0) \vee \\ \ldots \vee (f_1(p) \leq 0 \wedge \ldots \wedge f_{n-1}(p) \leq 0 \wedge f_n(p) = 0). \end{array}$$

In the most general case of a disjunction of conjunctions of inequalities,

$$c_j(p) \iff \begin{array}{l}(f_{11}(p) \leq 0 \wedge \ldots \wedge f_{1n_1}(p) \leq 0) \ \vee \ldots \\ \ldots \vee \ (f_{m1}(p) \leq 0 \wedge \ldots \wedge f_{mn_m}(p) \leq 0).\end{array}$$

The boundary is described by a disjunction of $(n_1 + \ldots + n_m)$ conjunctions, each block of conjunctions being obtained with the previous pattern (see Figure 4.3).

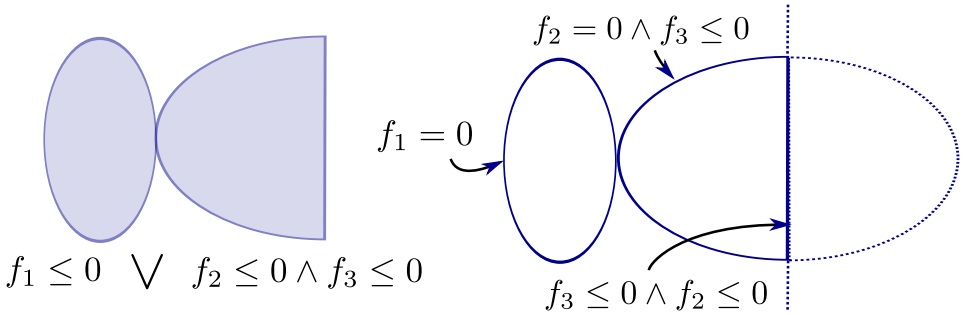


Figure 4.3 – **Boundary of an object.**

The boundary constraint c'_j can therefore be automatically generated by a simple symbolic processing.

The angle $\overline{\alpha}$ and $\underline{\alpha}$ are then two solutions of the system $c'_j(R_{-\alpha}(\tilde{p} - \tilde{o_j}))$ which is a disjunction of 1-dimensional equations (of variable α), i.e., something very easy to solve.

Note that if the shape is a disjunction with two subparts that plainly intersect (contrary to the figure), the boundary is only a subset of the generated system, in which case the angle inflation may not be optimal.

The algorithm of the inner inflation with respect to rotation is finally given below (see Listing 4.2).

Listing 4.2 – Inner inflator with respect to rotation

```
/**
 * \parameter c_j:      The moving object
 * \parameter q̃_j:      Initial configuration of the moving object (position and
 *                      rotation angle)
 * \parameter p̃:        Intersection point between the reference and moving objects
 * \parameter [α_j]:    Domain inside which inflation should be done (only the angle)
 *
 * \return               The inflated interval for α_j
*/
Interval inflate_rot(Object c_j, Point q̃_j = õ_j × α̃_j, Point p̃, Interval [α_j]) {

        // Domain to search the solutions of the 5-dimensional system of equations
        // We set the maximal possible domain for the angle
        Box box   = õ_j × [−π, π] × p̃

        Real lb = −∞
        Real ub = ∞

        // Find the smallest enclosure [lb, ub] for the initial angle α̃_j
        // among all the solutions
        for i from 1 to n_1 + ... + n_m { // for each conjunction in c'_j(p)
```

```
                // Solver for a 5-dimensional system of the form:
                // f_1(R_{-α}(p-o)) = 0 ∧ f_2(R_{-α}(p-o)) ≤ 0 ∧ ... ∧ f_n(R_{-α}(p-o)) ≤ 0.
                // Variables are o ∈ ℝ², α ∈ ℝ and p ∈ ℝ² (in this order).
                // All except α will be fixed.
                Solver solver(i^th conjunction of c'_j(p))

                // Find all the solutions inside a box
                vector<Box> sols = solver.solve(box)

                for j from 0 to sols.size() {

                        [α] = sols[j][2] // get the angle component

                        if (ᾱ > α̃_j) {
                                if (α̲ < α̃_j) {
                                        // impossible to build an interval
                                        // around α̃_j
                                        return [α̃_j, α̃_j]
                                } else if (α̲ < ub) { ub = α̲ }
                        }
                        else if (ᾱ > lb) { lb = ᾱ }

                }
        }

        if (lb = -∞) {

                if (ub ≠ ∞)
                     { [α_j] = [α_j] ∩ [-π, ub] }
                // Else, there is no intersection,
                // the object can rotate with any angle

        } else {
                if (ub = ∞)
                        { [α_j] = [α_j] ∩ [lb, π] }
                else
                        { [α_j] = [α_j] ∩ [lb, ub] }
        }

        return [α_j]
}
```

4.2.4 Global Inflation

We finally show that the two inflations can be combined, that is, the cartesian product of $[o_j]$ and $[\alpha_j] = [\underline{\alpha}, \overline{\alpha}]$ is a valid inflation of $\tilde{q_j}$.

The inflation w.r.t. translation produces a box $[o_j]$ that satisfies

$$\forall o_j \in [o_j], \quad c_i(o_j + (\tilde{p} - \tilde{o_j})).$$

The inflation w.r.t. rotation produces a box $[\alpha_j]$ that satisfies

$$\forall \alpha_j \in [\alpha_j], \quad c_j(R_{-\alpha_j}(\tilde{p} - \tilde{o_j})).$$

Together, they give

$$\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j], c_i(o_j + (\tilde{p} - \tilde{o_j})) \wedge c_j(R_{-\alpha_j}(\tilde{p} - \tilde{o_j})).$$

Substituting p to $(o_j + \tilde{p} - \tilde{o_j})$, we obtain:

$$\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ \exists p,\ c_i(p) \wedge c_j(R_{-\alpha_j}(p - o_j)),$$

that is, $\forall o_j \in [o_j],\ \forall \alpha_j \in [\alpha_j],\ (o_j, \alpha_j) \in \mathcal{S}_{ij}$.

The final algorithm is presented in Listing 4.3.

Listing 4.3 – Inner inflator for the overlapping region (inequality case)

```
/**
 * \parameter c_i: Reference object
 * \parameter c_j: Moving object
 * \parameter q̃_j:   Initial configuration of the moving object (position and
 *                   rotation angle)
 * \parameter [q_j]: Domain inside which õ_j and α̃_j have to be inflated
 * \parameter rot: <true|false> if the orientation is considered or not
 *
 * \return The inflated box for o_j and α_j
 */
Box inflate(Object c_i, Object c_j, Point q̃_j = (õ_j, α̃_j), Box [q_j] = [o_j] × [α_j], rot) {

        // Inflate q̃_j inside Object j
        if (rot) {
                // Inflate the angle
                [α_j]' = inflate_rot(c_j, q_j, p̃, [α_j])

                // cannot inflate the origin inside Object j
                [o_j]' = õ_j

        } else {
                // no angle (we assume the value is fixed to 0 in this case)
                [α_j]' = [0, 0]

                // Inflate the origin with respect to translation
                [o_j]' = inflate_trans(c_j, q_j, p̃, [o_j]) }

                if ([o_j]' = [o_j]) { // full inflate, useless to go further.
                        return [o_j]' × [α_j]'
                }
        }


        // Solver for a 5-dimensional system of the form c_i(p) ∧ c_j(R_{-α}(p - o)).
        // Variables are o ∈ ℝ², α ∈ ℝ and p ∈ ℝ² (in this order).
        // All except p will be fixed.
```

```
        Solver solver(c_i ∧ c_j)

        Box box = õ_j × α̃_j × ℝ²

        // Find intersection point. The solver stops after having found one
            solution.
        // If no solution is found, an error should be raised.

        Point (õ_j, α̃_j, p̃) = solver.find_solution(box)

        // Shift [o_j] to the intersection point, this gives
        // the domain inside which p̃ has to be inflated
        [p] = [o_j] + (p̃ - o_j)

        // Inflate p̃ inside Object i
        [p]' = inflate_trans(c_i, 0_ℝ³, p̃, [p])

        [o_j]' = [o_j] ∩ ([o_j]' + ([p]' + o_j - p̃))  // sum of boxes (<=> Minkowski sum)

        return [o_j]' × [α_j]'
}
```

4.3 Using Contact Conditions

There exists a complete different approach for calculating the overlapping region (or directly evaluating the overlapping function) which also makes sense in the case of objects described by inequalities. It consists in enforcing the so-called *contact conditions*. This approach is very appealing for different reasons to be explained soon, but it raises also some new difficulties that we could not really manage to cope with. So, it has finally been discarded in the course of this PhD, but only after a long delay.

The principle is the following. Let us assume first that each object is described by a single inequality, that is,

$$\begin{aligned} c_i(p) &\iff f_i(p) \leq 0, \\ c_j(p) &\iff f_j(p) \leq 0, \end{aligned}$$

with f_i and f_j everywhere differentiable.

We notice now that, given a fixed configuration $q_i = 0$ and whatever is the initial configuration for Object j, its closest configuration that makes the two objects non overlap is necessarily reached at a point $q_j = (o_j, \alpha_j)$ that makes the objects touch tangentially (see the left side of Figure 4.4), tangent objects being considered here as non-overlapping, of course. This means, first, that there exists at least one point p that lies at their respective boundaries or, in other words, that the equalities

$$f_i(p) = 0 \quad \text{and} \quad f_j(R_{-\alpha_j}(p - o_j)) = 0$$

are actually satisfied.

Second, the normal vectors of each curve have to be collinear with opposite orientation, as depicted in Figure 4.4.

The normal vector of the implicit curve $f(x) = 0$ which is oriented towards the positive values (i.e., towards the exterior of the object) is simply the gradient vector $\nabla f(x)$. The gradient of $f(x)$ is the vector of partial derivatives with respect to each component of x, that is,

$$\nabla f(x) = \Big(\frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_d}(x)\Big).$$

So this condition can be enforced by constraining the determinant of the two gradients to be null and their scalar product to be negative.

Hence, the boundary of the overlapping region $\mathcal{S}_{ij}$, which we denote by $\partial\mathcal{S}_{ij}$, satisfies:

$$q_j \in \partial\mathcal{S}_{ij} \Longrightarrow \begin{cases} f_i(p) = 0 \\ f_j(R_{-\alpha_j}(p - o_j)) = 0 \\ \det(\nabla f_i(p), \nabla f_j(R_{-\alpha_j}(p - o_j)) = 0. \\ \nabla f_i(p) \cdot \nabla f_j(R_{-\alpha_j}(p - o_j) \leq 0. \end{cases} \tag{4.1}$$

Similarly, in order to evaluate the overlapping function, one can solve the following problem:

$$\text{minimize } \|q'_j - q_j\| \quad \text{subject to} \quad \begin{cases} f_i(R_{-\alpha_i}(p - o_i)) = 0 \\ f_j(R_{-\alpha'_j}(p - o'_j) = 0 \\ \det(\nabla f_i(R_{-\alpha_i}(p - o_i)), \nabla f_j(R_{-\alpha'_j}(p - o'_j)) = 0. \\ \nabla f_i(R_{-\alpha_i}(p - o_i)) \cdot \nabla f_j(R_{-\alpha'_j}(p - o'_j) \leq 0. \end{cases} \tag{4.2}$$

Unfortunately, only the implication is true in (4.1), the converse relation is wrong. Some configurations q_j may satisfy the contact conditions while being in the interior of $\mathcal{S}_{ij}$. Such situation is depicted on the right side of Figure 4.4. This means that the minimum for (4.2) is only a lower bound of the true value of the overlapping function. The contact conditions actually describes a larger set than $\partial\mathcal{S}_{ij}$, called the *weak boundary* and which is the union of the real boundary and *fake* boundaries.

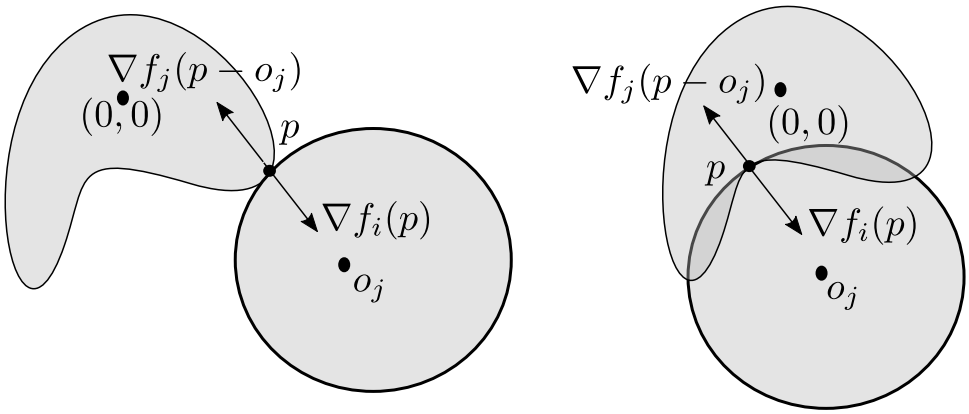


Figure 4.4 – **Contact Conditions.** *Left:* A configuration q_j that satisfies the contact conditions. *Right:* Another configuration for the same object that also fulfills the contact conditions and which produces a fake boundary.

So these conditions do not solve the problem by their own. We have therefore tried in many different ways to take advantage of these conditions by incorporating them into our existing inflators and rejection test. We expected a significant speed-up but the experiments were quite disapointing. Unfortunately, we did not keep a clean record of them but the conclusion was that removing all the fake boundaries is an as difficult problem as calculating the boundary itself. So we finally drop the use of these conditions. This conclusion is also an echo of our discussion in Section 1.3.2.

4.4 Experimental Results

The goal of these preliminary experiments is to validate the theory and to have a first impression of how the algorithm behaves in practice. The ambition is neither to solve real-world problems nor to make an extensive performance analysis.

In the first part of this section, we start by comparing the performances of our paving algorithm with RSOLVER, a generic state-of-the-art solver for continuous quantified constraints. In the second part, we compare the inflator of the previous chapter (the one based on the Minkowski sum) with that of the present chapter.

Various packing instances are then solved in the last part. The difficulty of a packing instance, besides the number of objects of course, mainly depends on the following criteria:

— *variable occurrences*: the number of times each variable appears in the expressions of the inequalities;
— *convexity*: if objects are non-convex, the boundary of the non-overlapping constraint will be less smooth. So the paving will be more complicated (hence more time consuming), especially near the boundary;
— *degrees of freedom*: that is, whether we take into account rotation or not. Allowing rotation gives a problem of much higher difficulty for multiple reasons. First, the size of the paving is exponential in the number of degrees of freedom so we cannot expect to get a 3D paving within the time scale of a 2D paving. Second, the angles in the inequalities create a lot of multi-occurrences and considerably increases non-convexity by the introduction of trigonometric functions.

4.4.1 Comparing with Quantified Constraints Solving

Remember that the overlapping constraint can be cast as a quantified constraint (see §1.3.2). We first compare one of our strategies with the use of RSOLVER, a generic solver for quantified constraints. Note that the latter interprets the precision ε set by the user in a different way than us. The paving is stopped when the area of the boundary region is less than ε times the area of the initial box. So, we have adapted our algorithm to make a fair comparison with RSOLVER.

We compare now the pavings calculated with three objects of increasing complexity: an ellipse, a rotated ellipse and a *peanut* (see Figure 4.5).

Table 4.1 reports the running times obtained by RSOLVER and by the Minkowski sum approach. The precision ε was set each time to 3.25% and the rotation angle was set in $[0, 0.7]$ for the ellipse-ellipse case,

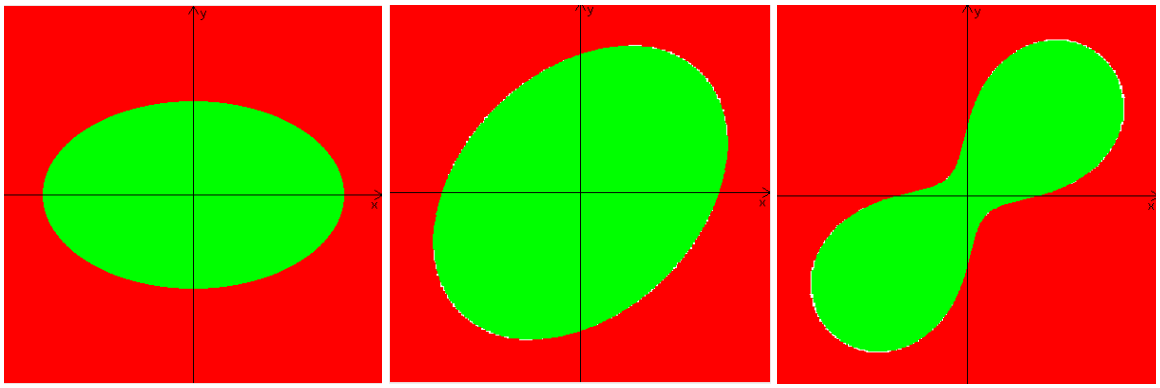


Figure 4.5 – **Objects to compare with the Minkowski sum approach.** From left to right: ellipse, rotated ellipse and peanut.

and $[0, 0.3]$ for the peanut-peanut case. The choice for ε corresponds to the minimal value that gives no timeout with RSOLVER.

Reference object	Moving object	Rsolver	Minkowski sum approach
Ellipse	Ellipse	4.07	0.37
Ellipse	Rotated Ellipse	51.55	2.67
Ellipse	Peanut	611.85	7.90
Rotated Ellipse	Rotated Ellipse	132.46	5.58
Rotated Ellipse	Peanut	656.11	12.00
Peanut	Peanut	771.00	26.82
Rotated Ellipse	Rotated Ellipse	4008.00	540.00
Peanut rotated	Peanut rotated	> 6000.00	> 6000.00

Table 4.1 – **RSOLVER vs Minkowski sum approach.** Running time (in s) for the 6 translation cases and the 2 rotation cases (with $\varepsilon = 3.25\%$).

Our algorithm outperforms RSOLVER. However, one should keep in mind that RSOLVER is a generic solver for quantified constraints, while our algorithm is dedicated to a particular class of quantified constraints (namely, overlapping constraints).

4.4.2 Comparison with the Minkowski Sum Approach

We compare now the inflation algorithm proposed in this chapter and the inflation algorithm based on the Minkowski sum. Comparison is made with the same objects as previously and, again, for different values of the precision criterion ε. Furthermore, each algorithm comes in two flavors:

— a bisect & inflate strategy

— a bisect & contract strategy, by embedding the inflator inside a sweep loop (see §2.6)

So, the following experiments will compare both inflators, each time with and without a sweep-based contraction.

Note that, to avoid implementation issues, ε is defined now as the minimal width of a box to be bisected. It is clear that a fair comparison would rather require to set ε as before; indeed, nothing can be said about the *quality* of the pavings; a paving with a very thick boundary obtained quickly cannot be compared with a paving with a thin boundary obtained slowly. And the minimal width of a boundary box has nothing to do with the area of the whole boundary.

The value of ε has been set in the translation case to 0.005 and in the rotation case to 0.0325. The domain of the rotation angle is now the interval $[-\pi, \pi]$.

Reference object	Moving object	M + S	M + I	C + S	C + I
Ellipse	Ellipse	7.05	3.36	5.63	3.63
Ellipse	Rotated Ellipse	14.54	18.42	16.61	15.89
Ellipse	Peanut	21.47	56.48	30.98	44.61
Rotated Ellipse	Rotated Ellipse	8.88	10	8.12	8.57
Rotated Ellipse	Peanut	26.54	29.94	11.29	22.86
Peanut	Peanut	36.05	74.53	22.67	54.31
Ellipse rotated	Ellipse rotated	3070.54	3235.98	10665.00	2301.24
Peanut rotated	Peanut rotated	8564.00	8531.00	24813.00	4709.00

Table 4.2 – **Minkowski sum approach vs Current algorithm.** Running time (in s) of the Minkowski sum approach with sweep (M+S) , or with inflation only (M+I), and the current algorithm with sweep (C+S) or inflation only (C+I).

Globally, the current algorithm using only the inflation gives better results than the Minkowski sum approach with or without sweep.

4.4.3 Packing Setup

We have created 5 cases of increasing difficulty. In each case, n objects have to be placed with $n = 10$, 20 and 30. Each time, the size of the container has been adjusted manually so that the density of the resulting packing looks similar.

— **Case 1.** This is a standard circle packing problem. The objectives are, first, to check that our algorithm also succeeds in solving this problem and, second, to measure its overhead as compared to a dedicated approach. In this circle packing instance, the radius of the i^{th} circle is equal to $\sqrt{i}$, $1 \leq i \leq n$. The radius of the enclosing circle is set to 2.1 ($n=10$), 2.35 ($n=20$) and 2.48 ($n=30$).

— **Case 2.** The purpose of this case is to introduce a new shape, that is, for which (to our knowledge) no dedicated solver exists. The problem is otherwise kept as simple as possible: (1) there is only one shape in addition to the enclosing shape, (2) this shape is described by a single inequality and (3) only translation is permitted (no rotation). The shape we have chosen is an ellipse of radius 1 and 0.5. The ellipses have to be packed inside a circle of radius 2.9 ($n=10$), 4 ($n=20$) and 5 ($n=30$).

— **Case 3.** We increase the complexity by allowing rotation in the same ellipse packing instance. With this new degree of freedom, the enclosing circle can be chosen a little bit smaller. The radius of the

circle is set to 2.7 ($n=10$), 3.9 ($n=20$) and 4.9 ($n=30$).

— **Case 4.** We increase again the complexity by replacing the ellipse with a non-convex shape which description involves several inequalities. The shape is the horseshoe depicted in Figure 1.3. The enclosing shape is an ellipse of radius 2/4 ($n=10$), 3.9/1.8 ($n=20$) and 5/2.9 ($n=30$).

— **Case 5.** The last case is a mixed packing problem where n/2 ellipses and n/2 horseshoes have to be packed inside a circle of radius 2.9 ($n=10$), 4 ($n=20$) and 6 ($n=30$). Again, both translation and rotation are considered.

4.4.4 Paving Results

Since the same pavings can be used in different cases, paving and packing times are reported separately. We also only give the average time for the 13×12 circle-circle pavings calculated in Case 1 and for the other pavings involving the enclosing shape (since this shape changes with n). Tables 4.3 and 4.4 summarize the paving and packing times. Figures 4.6, 4.7 and 4.8 shows pictures of the results. Every paving has been calculated with a precision ε set to 0.1.

d.o.f.	*Object i*	*Object j*	*Time*
translation	circle	circle	0.01 (avg)
	ellipse	ellipse	0.16
	enclosing circle	ellipse	0.81 (avg)
translation and rotation	ellipse	ellipse	300
	enclosing circle	ellipse	2245 (avg)
	ellipse	horseshoe	1740
	horseshoe	horseshoe	3780
	enclosing circle	horseshoe	8311 (avg)
	enclosing ellipse	horseshoe	10909 (avg)

Table 4.3 – **Paving time** (in seconds)

4.4.5 Packing Results

In the packing results table we can see that the circle packing problem has been solved in a few seconds only. However, the enclosing circle is not optimal. In [Martinez 2013], the circles are packed in a circle of radius 1.95 ($n = 10$), 2.15 ($n = 20$) and 2.27 ($n = 30$). This gap is due to the precision ε. Using this precision for paving amounts to consider that objects to pack are *enlarged by* ε. The radius we have set for the enclosing circle is actually the smallest one for which packing is successful, given the precision ε of 0.1.

Packing for the other cases has been done in the order of the minute. Above all, we see in Figure 4.8 that the holes introduced by non-convex objects like the horseshoe are not lost. The paving time is in the order of one hour.

#objects	case 1	case 2	case 3	case 4	case 5
10	<0.1	4.5	66	103	53
20	54	32	235	372	194
30	203	53	559	1005	395

Table 4.4 – **Packing time** (in seconds)

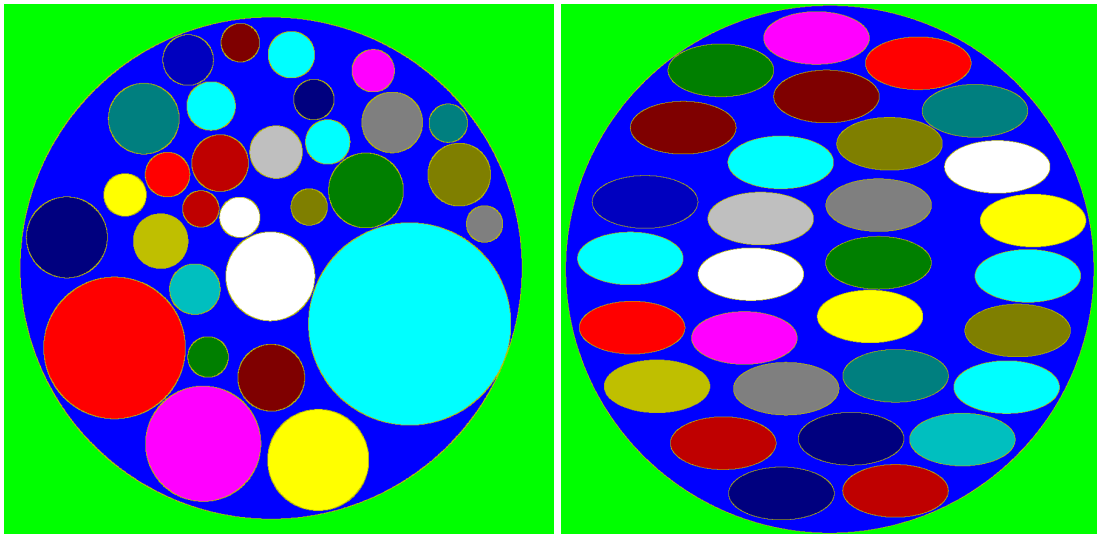


Figure 4.6 – **Packing results. (Left)** Case 1. **(Right)** Case 2.

4.5 Conclusion

Our packing framework requires the pre-calculation of the overlapping region. In this chapter, we have introduced a method to compute a paving of the overlapping region when the objects are described by inequalities. The main component of the paving algorithm being the inner inflation, we have focused on this point.

The inflator proposed here is split in two steps: inflation w.r.t. translation and inflation w.r.t. rotation. The former is very similar to the inflator proposed in the last chapter as it is simply based on the Minkowski sum equivalence, though it is not really necessary to think in these terms. The latter handles optimally the angle inflation, which is its main advantage. Instead of trying to inflate the position and the orientation at the same time, we fix now the orientation when inflating the position and conversely. More precisely, we inflate the orientation by computing the two nearest values for the angle, such that the boundary of Object j meets an inner point of Object j that belongs to Object i for all the positions obtained for the origin when inflating w.r.t. translation. Consequently, if the result of the inflation w.r.t. translation is the box $[o_j]$ and the result of the inflation w.r.t. rotation is the interval $[\underline{\alpha_j}, \overline{\alpha_j}]$, then, the final inflation is simply the Cartesian product $[o_j] \times [\underline{\alpha_j}, \overline{\alpha_j}]$.

In this chapter, we have also made an important digression about *contact conditions*. This approach is often used in the collision detection field, but was finally discarded for efficiency reasons.

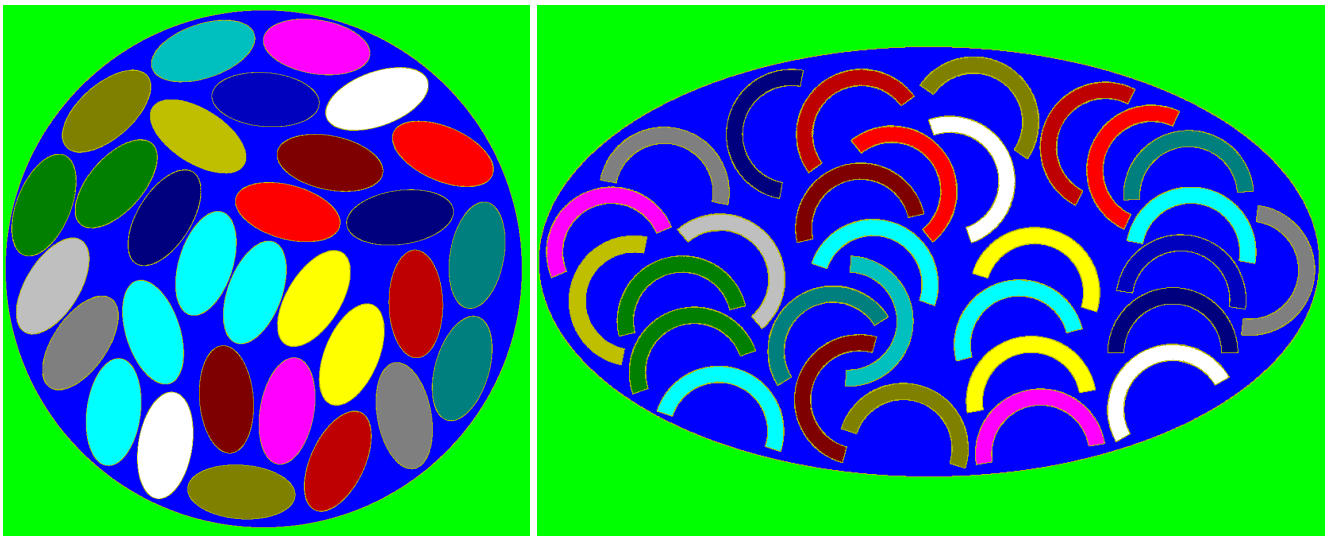


Figure 4.7 – **Packing results. (Left)** Case 3. **(Right)** Case 4.

We have finally provided preliminary experimental results.

In the first experiments, we compare the ability of the Minkowski sum approach to describe the overlapping region with a state-of-the-art quantified constraint solver called RSOLVER. The results obtained show that our proposal computes the paving of the overlapping region in significantly less time, which justifies the recourse to a dedicated approach.

The second experiments still focus on the overlapping region and compare the peformance of the inflator proposed in Chapter 3 with the two-steps inflator proposed in this chapter. Different variants are considered, depending if the paving algorithm is based on a bisect & contract approach (with the sweep algorithm) or the more simple bisect & inflate one. Globally, the combination of the two-steps inflator with the sweep algorithm gives the best times.

Finally, in the third experiments, we test the overall framework on packing problems. As expected, calculating the pavings of the overlapping regions for all pair of shapes can take a considerable amount of time, but these pavings are only calculated once whatever is the actual number of objects. Packing up to 30 objects has been done in the order of the minute. It can be observed that the placement of objects fill the spaces produced by the non-convexity of shapes.

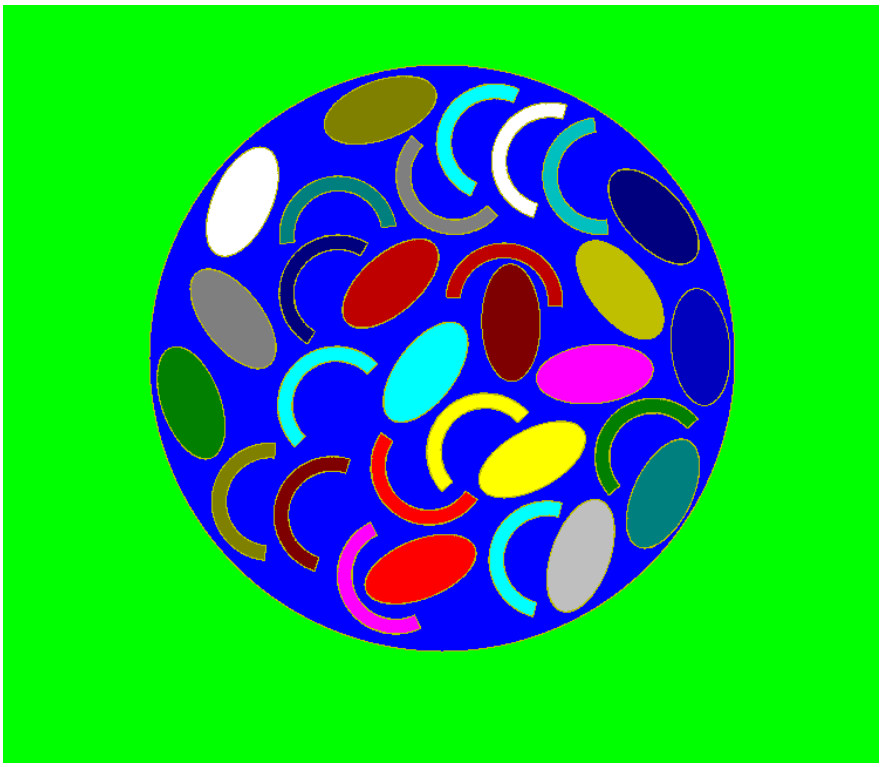


Figure 4.8 – **Packing results, case 5.**

5

Object Described by Parametric Curves

5.1 Introduction

The purpose of this last chapter is to instantiate our general packing strategy to the case of objects described by parametric curves.

A parametric curve is the image of an interval by a vector-valued mapping. Formally, given a mapping

$$f : \mathbb{R} \to \mathbb{R}^2$$

we call parametric curve, the set

$$\mathcal{C} := \{f(t), t \in [0, 1]\}.$$

The *orientation* of the curve (that is, whether the curve is described from one vertex to another as t moves from 0 to 1, or in the other way round) has no incidence on the proposed algorithms.

Parametric curves are intensively used in computer-aided design and, in particular, Bézier curves and their extensions (B-splines, etc.) [Shah 1995, Snyder 2014].

5.1.1 Bézier curves

A Bézier curve [Jolly 2009, Kirk 2012] is a polynomial curve of the following form

$$b(t) = \sum b_i B_i^n(t), \quad t \in [0, 1]$$

where the b_i's are given vectors of $\mathbb{R}^2$ called *control points* and B_i^n are so-called *Bernstein polynomials* [Carnicer 1993, Farouki 2012]:

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}.$$

The convex hull of the control points is also a polygon that encloses the Bézier curve (see Figure 5.1). Hence, Bézier curves can be easily enclosed into polygons. This polygon is called the *convex hull* of the curve.

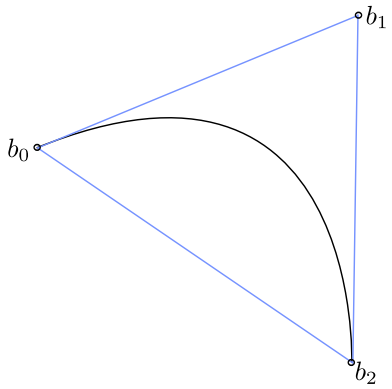


Figure 5.1 – **Convex Hull of a Bézier Curve**. The vertices b_0, b_1 and b_2 of the convex hull are control points of the curves.

We assume from now on that an object is defined as the area inside a close chain of parametric curves, as depicted in Fig. 5.2.

5.1.2 Bézigons

Contrary to inequalities where the constraint associated to an object was obtained by means of logical operations (intersection and union), objects described by parametric curves are characterized by a *list* of curves, each curve corresponding to a piece of its boundary, and it is assumed that each piece is connected to another at its two endpoints and that the pieces altogether form a closed and non self-intersecting loop, that is, the object is well-formed (see Figure 5.2). When curves are Bézier curves, such object is often called a *bézigon*.

To instantiate our packing strategy to the case of parametric curves, we will keep in line with the previous chapter and try to see how the two-steps inflators To this end, we are simply going to revisit the different components of this inflator which were specific to inequalities.

However, polyhedral approximations seem also to be particularly well-suited in the context of parametric curves and we shall start this chapter by explaining how they could indeed be used and why, in spite of this, we have not adopted this approach.

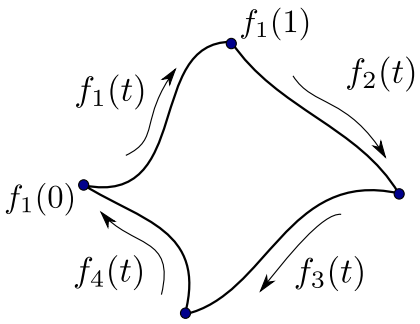


Figure 5.2 – **Object described by parametric curves**

5.2 Polyhedral approximation

Parametric curves are suitable to polyhedral approximation. This is actually true for *any* nonlinear parametric curves as one can build numerically a rigorous polyhedral approximation of a curve thanks to the interval enclosure of derivative vectors, but this is even more true for Bézier curve where such an enclosure can be obtained without resorting to interval techniques (except for handling roundoff errors). This is due to a *sub-divisibility* property of Bézier curves.

A Bézier curve can indeed be divided into two sub-Bézier curves, and so on. This means that a Bézier curve can be split into an arbitrary number of small portions. For each portion of the curve, a convex hull can be obtained from its control point, and the union of all these polygons gives then an accurate polyhedral approximation of the curve (see Figure 5.4).

To divide a Bézier curve into two Bézier curves, we make use of De Casteljau's algorithm [Boehm 1999, Jain 2012]. This algorithm allows to evaluate $b(t)$ for a given t in a different manner than with Bernstein polynomials. But this algorithm also allows to split the curve in two portions:

$$\tau \in [0,t] \mapsto b(\tau) \quad \text{and} \quad \tau \in [t,1] \mapsto b(\tau).$$

Definition 11 (De Casteljau's algorithm)
Given a Bézier curve $b(t)$ described by the control points $b_0, ..., b_n$, the De Casteljau's algorithm calculates the subdivision points $b_i^{(r)}$ iterating the following expression:

$$b_i^{(r)} := b_i^{(r-1)}(1-t) + b_{i+1}^{(r-1)}t, \tag{5.1}$$

where r is the iteration number and $b_i^{(0)} := b_i$.

The iteration continues until there is only one subdivision point and this point satisfies $b_0^{(n)} = b(t)$.

It turns out that the algorithm also generates the control points of the two sub-Bézier curves (see Figure 5.3), which are respectively $\{b_0, b_0^{(1)}, \ldots, b_0^{(n)}\}$ and $\{b_0^{(n)}, b_1^{(n-1)}, \ldots, b_n\}$. The successive subdivisions produces a set of Bézier curves where their convex hulls encloses better the Bézier curve. The union of the two

portions hulls produces a more accurate approximation of the initial curve than its hull. This process can be repeated recursively.

The union of all the portions hulls of a curve gives then an outer approximation of that curve, and the union of all the curves approximations of a bézigon gives an outer approximation of its boundary. This may look as a complicate task. In fact, we don't need to calculate this. We simply take the hull of all the control points given by the previous division method, applied on each curve of the bézigon. This immediately gives an outer approximation of the bézigon. The convex hull of a set of points is a standard problem in computational geometry and we have tested this approach using the state-of-the-art library CGAL [Fabri 1998].

Now that objects are polygons, we can calculate the overlapping region by calculating the Minkowski sum of polygons. The Minkowski sum of convex polygons is also a standard problem in computational geometry. It can also be extended to non-convex polygons with two strategies, the first being based on *decomposition* and the second on *convolution*.

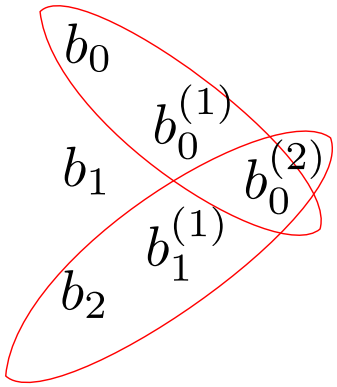


Figure 5.3 – **De Casteljau's Iterations**. The Bézier curve is split into two sub-curves with common point $b_0^{(2)}$.

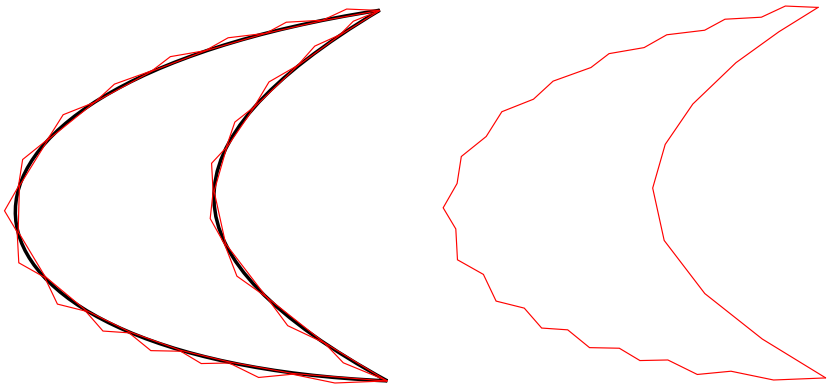


Figure 5.4 – **Polyhedral Approximation of the half- moon**. *(Left)* Curve Subdivision. *(Right)* Polyhedral approximation of the object.

In the first strategy, two non-convex polygons P and Q are decomposed into two sets of convex polygons

$\{P_1, ..., P_k\}$ and $\{Q_1, ..., Q_t\}$, where the union of the polygons of each set is equal to the initial polygon:

$$\bigcup_{i=1}^{k} P_i = P \quad \text{and} \quad \bigcup_{j=1}^{t} Q_j = Q.$$

The Minkowski sum is then performed over each pair of sub-polygons, $S_{ij} = P_i + Q_j$. Finally the Minkowski sum is the union of the Minkowski sums S_{ij},

$$P + Q = \bigcup_{ij} S_{ij}.$$

The second strategy considers the set of vertices $(p_0, p_1, \ldots)$ and $(q_0, q_1, \ldots)$ of the two polygons P and Q, and calculates all the segments of the form

$$[p_i + q_j, p_{i+1} + q_j] \quad \text{and} \quad [p_i + q_j, p_i + q_{j+1}].$$

This is called the *convolution* of P and Q. These segments form a number of closed polygonal curves called convolution cycles. The Minkowski sum $P + Q$ is then the polygon form by the points that have a non-zero *winding number* with respect to these cycles (this technique based on the winding number basically eliminates the *fake boundaries* we have discussed about in §4.3).

The number of segments in the convolution of two polygons is usually smaller than the number of segments that constitute the boundaries of the sub-sums S_{ij} using the decomposition approach. As both approaches construct the arrangement of these segments and extract the sum from this arrangement, computing the Minkowski sum using the convolution approach is usually faster.

CGAL can handle the Minkowski sum between two non-convex polygons using both approaches. The results we have obtained using a polyhedral approximation are impressive, thanks to CGAL. For a given required precision on the Minkowski sum surface, the computation times with this approach are incomparably better than the ones with our inflate-and-bisect strategy.

However, this approach is limited to the translation of object. Introducing the augmented object (see 3.3.3) to cast again the problem into a Minkowsk sum creates another big difficulty: the object are three-dimensional and their parametric surfaces are not polynomial anymore. Calculating a rigorous mesh of such objects is a difficult problem on its own.

So, polyhedral approximations is definitely the best approach for packing bézigons that can be translated. But if rotation is considered, our framework is still a valuable option.

5.3 Position of a Point w.r.t. an Object

Let us now get back to our framework. As said, we need to extend the inner inflator and the outer contractor to the case of parametric curves.

The very first step of the inflator consists in checking if a point $\tilde{p}$ is inside the object. The difficulty of this test is the most blatant difference with the implicit case (inequalities).

Let us start with the simple case of the reference object i (no translation nor rotation), characterized by a single function f.

Remember that in the implicit case, the constraint describing an object was simply:

$$c_i(p) \iff f(p) \leq 0. \tag{5.2}$$

and this constraint is precisely the membership test for the reference object.

In the parametric case, there is no such direct mathematical formula. We have to count the number of times an (arbitrary chosen) half-line emanating from p crosses the boundary of the object [1]. The point is then inside the object if and only if this number is odd (cf. Figure 5.5).

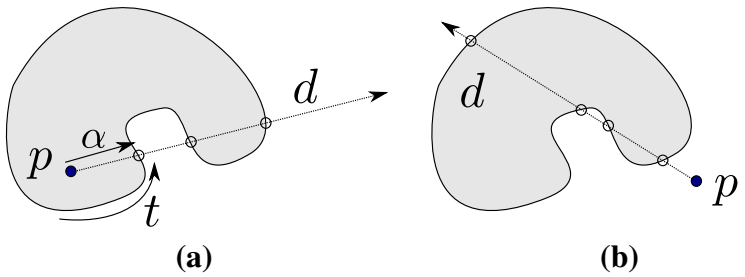


Figure 5.5 – **Membership test**. **(a)** the half-line intersects an odd number of times the boundary of the object (here, 3), so the point p is inside. **(b)** the number of intersection is even (here, 4), the point is outside.

Denoting d the direction of this half line, we obtain the following characterization, which is far less straightforward:

$$\begin{aligned} c_i(p) &\iff s(f,p) \mod 2 = 1 \\ \text{with} \quad s(f,p) &= \Big|\{\alpha > 0,\ \exists t \in [0,1],\ f(t) = p + \alpha d\}\Big|. \end{aligned} \tag{5.3}$$

In the case of the container, which is the complementary of an object, the inner test corresponds to an even number of intersection.

Note that the half-line d can be chosen randomly but there are bad choices that lead to singularities (non-invertible systems $f(t) = p + \alpha d$). This corresponds to the case of a line crossing the boundary tangentially in at least one point. By the way, the problem appears numerically when the system to be solved is *close* to a singularity (while not being really singular, in theory). The good news is that the interval Newton operator (see 2.4) can detect this problem as boxes get closer to the singular solution: the interval evaluation of the jacobian matrix is non-invertible (only contains singular matrices) and this can be assessed. In this situation, we simply have to chose another direction.

In theory, this process has to be repeated until a "good" direction is found but, in practice, a second attempt is enough (the probability of choosing consecutively two directions that cross the boundary tangentially is quasi-null).

1. In the case of a polygon, we could also use the *argument pinciple*. But a polygon, even non-convex, can be obtained as a union of polytopes; so this is an object which admits an implicit representation.

Back to our inflator, the point $\tilde{p}$ has to belong to both objects: the reference one and the moving one. In the case of the moving object j, we have to take into account its configuration.

The system to be solved inside (5.3) is not anymore

$$f(t) = p + \alpha d$$

but

$$o_j + R_{\alpha_j} f(t) = p + \alpha d.$$

Finally, by noting $c_j(q_j)(\cdot)$ the membership test for an object with configuration q_j and $s_j(q_j, p, d)$ the number of intersection point between this object and the half-line of direction d emanating from p, we have:

$$c_j(q_j)(p) \iff s(o_j + R_{\alpha_j} f,\ p) \mod 2 = 1,$$

where, remember,

$$c_j(q_j)(p) \iff c_j(R_{-\alpha_j}(p - o_j)).$$

Finally, in the case of an object with multiple curves $f_1, \ldots, f_n$, which is what happens usually (like in Figure 5.2) we have to sum up the number of solutions obtained with each curve, that is,

$$c_j(q_j)(p) \iff \Big(s(o_j + R_{\alpha_j} f_1, p) + \ldots + s(o_j + R_{\alpha_j} f_n, p)\Big) \mod 2 = 1.$$

5.4 Status of a Box w.r.t. an Object

The second component which was specific to inequalities is the inflation of the point $\tilde{p}$ with respect to an object. We will not propose an inflator here but something simpler, namely an *inclusion test*. It will be explained further how the strategy can be enforced without inflator.

The question is more precisely to detect the status of a box with respect to an object. Three cases are actually possible:

1. the box is crossing the object
2. the box is inside the object,
3. the object is inside the box.

Indeed, since the box contains a point $\tilde{p}$ which is already proven to be inside, the box and the object cannot be disjoint.

In the implicit case, the inequality (5.2) can easily be transformed into an inclusion test; this is a classical result of interval analysis. But Formula (5.3) requires a specific algorithm and this is the purpose of this section.

First of all, notice that Relation (5.3) cannot be easily *intervalized*, in the sense that computing an interval $[s]$ enclosing the number of intersection points obtained in a given direction d, or,

$$[s] \supseteq \{s(q_i, p),\ p \in [p]\}$$

raises a number of difficulties.

First, counting the number of solutions of a system of equations with thick parameters (the coordinates of p, which are now ranging within a non-degenerated box $[p]$) cannot always be guaranteed although there is no singularity at all. This typically happens in the case where the distance between two solutions is smaller than the uncertainty on parameters: the Newton operator cannot isolate each solution (which is mandatory for certifying each) because of the pessimism introduced by the parameters.

Above all, an interval $[s]$ like $[0, 2]$ does not allow to decide whether the box is outer or not (because the value 1 can be taken or not), though that should only be a problem for boxes $[p]$ larger than a certain size.

A much better alternative consists in focusing on the contour of the box $[p]$. However, we need to assume that objects have no *hole*. Formally, an object has no hole if the complementary of the object is a connected set.

The principle is then the following. It is easy to check if two curves intersect using an interval solver. So we can easily check if the contour of the box intersects the boundary of the object, by considering the 4 segments of the contour and each curve of the object. If one segment intersects the object boundary then we are in Case 1 (the box is crossing).

If no segment intersects the object the two other cases are possible. To settle, we consider a point p on the contour of the box $[p]$ (e.g., a vertex) and test if this point is inside the object or not using §5.4. If it is inside, we are in Case (2) because we have assumed that the objet has no hole. Otherwise, we are in Case (3). We can also apply another trick. We can trace a 5^{th} segment joining $\tilde{p}$ to the contour of the box and verify if this new segment intersects the object. If it does, we are in Case (3) (cf. Figure 5.6.a), otherwise we are in Case (2) (cf. Figure 5.6.b).

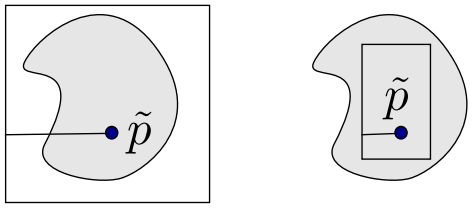


Figure 5.6 – **Inner test for the box**

5.5 Paving of the Overlapping Region

We describe here the inner and outer tests, given a moving object j and a reference object i, following the strategy already used in the previous chapters.

For the inner test, the strategy consists in splitting the inflation following the two transformations (translation/rotation). We present these two operations then we will explain why the inner test cannot be expanded (at least, easily) to make an inflator, in the implicit case.

5.5.1 Inflation with respect to translation

The first step of the inflation is to find a point $\tilde{p}$ at the intersection of the two objects. For that, we resort to a sub-solver, as in the implicit case. This sub-solver considers an initial box $[p] = \mathbb{R}^2$ which is bisected recursively and applies an inner test and an outer test based on the technics of §5.4. More precisely, given the current box $[p]$, the solver rejects the box if the box $[p]$ turns to be either outside Object i (with configuration 0) or outside Object j (with configuration $\tilde{q}_j$).

Otherwise, it takes a random point $\tilde{p}$ inside the box $[p]$ and applies the inner test on $\tilde{p}$ with both objects, the configuration of Object i (resp. j) being as before 0 (resp. $\tilde{q}_j$). If both tests are satisfied, the subsolver stops and $\tilde{p}$ is the sought point.

Once the point $\tilde{p}$ is found, we must inflate it to construct a box inside the object i.

Remember that the inflation, in the implicit case, was based on the inner arithmetic, see §4.2. The inflation in the parametric case is far from being trivial. An attempt, that we have long explored, is to use curve arrangement techniques, but the analysis of cases is prohibitive.

Another considered approach was to make use of the inner paving of the objects (see §3.3.7). The question amounts then to searching for a maximal box (in the sense of inclusion) which is included in a set of boxes (the inner sub-paving of the object considered) and which contains a given point (the intersection point $\tilde{p}$). However, formalizing correctly this problem requires to precisely define which maximal box is actually looked for, like, for instance, the box with maximal surface. There is no simple solution to this problem.

The chosen strategy is the following. Instead of calculating first a box $[p]$ inside the object, and then inflating inside $[o]_j$ as follows (see §2.7.2 for the notations):

$$In(\tilde{o}_j, [o]_j) := [o]_j \cap ([p] - \tilde{p} + \tilde{o}_j);$$

the idea is to make the contrary, that is, to translate the box $[o]_j$ and to verify that it is included in the object, that is, to check if

$$\forall p \in \Big([o]_j + \tilde{p} - \tilde{o}_j\Big) \quad c_i(0, p).$$

If the test returns $true$, we have proven that $[o]_j$ was inside the overlapping region.

This test is less powerful than an inflator, but is enough for generating the paving.

5.5.2 Inflation with respect to rotation

Contrary to the translation, the inflation with respect to rotation that we have proposed in the previous chapter can be easily carried over parametric curves, merely because the inflation only relies on the ability to solve equations that describe the object boundary, which is exactly what the parametric curves do.

5.5.3 Outer test

Not surprisingly, the outer test is also based on boundary intersection. We just check that no curve of the boundary of Object i intersects a curve of the boundary of Object j, when $q_j \in [q_j]$.

Assume for simplicity that Object i is described by one curve $f(t)$ and Object j by one curve $g(u)$, with $t, u \in [0,1]$.

We have to check that

$$\forall (o_j, \alpha_j) \in [q_j], \forall t \in [0,1],\ \forall u \in [0,1], \quad f(t) \neq o_j + R_{\alpha_j} g(u). \tag{5.4}$$

Again, this can be easily performed by a sub-solver that tries to enforce the equality. If the solver returns no solution, the test is passed. As before, the box $[q_j]$ should not be bisected by this sub-solver as this bisection would be redundant with that of the overall paving process. The subdivision is performed only for the variables t and u.

Of course, to be sure that the situation where (5.4) holds corresponds to disjoint objects (and not to Object j included inside Object i), we also have to fix the configuration of Object j to some arbitrarily q_j inside $[q_j]$, then to take an arbitrary point in the object (e.g., $g(0)$) and finally to check if

$$c_i(o_j + R_{\alpha_j} g(0)).$$

If the answer is "no", $[q_j]$ is outside.

Note that if the answer is "yes", then $[q_j]$ is inside the overlapping region. However this inner test is not interesting as it enforces an *inclusion* condition

$$\forall q_j \in [q_j],\ c_j(q_j)(p) \Longrightarrow c_i(p)$$

whereas an *overlapping* condition, which is much weaker, is enough for proving that $[q_j]$ is inner:

$$\forall q_j \in [q_j],\ \exists p \in \mathbb{R}^2, \quad c_j(q_j)(p) \wedge c_i(p).$$

5.6 Preliminary results

As in Chapter 4, we present in this section the times we have obtained for calculating a paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ of the overlapping region for every combination of object types, and the times for packing a set of objects using these pavings. Again, the objective of these experiments is to validate the theory and show the applicability of the approach, rather than providing a complete performance analysis.

We consider two types of objects to be packed: a *half-moon* and a *flower* (see Figure 5.7). Both objects are described by more than one quadratic curve and have different number of cavities. The number of curves has an impact on the paving time, since it makes the boundary of an object longer to be processed. The number of cavities has also an impact on the paving time, since it imposes more bisections to correctly describe the non-convex parts of the overlapping region, and an impact on the paving time, since it makes the overlapping function more "nonlinear".

Both the number of curves and cavities makes the flower a much more complicated object than the half-moon (the flower has 10 curves forming 5 cavities while the half-moon has only 2 curves and 1 cavity). That is why, at the current state of our developments, we are unfortunately only able to allow rotation for half-moons.

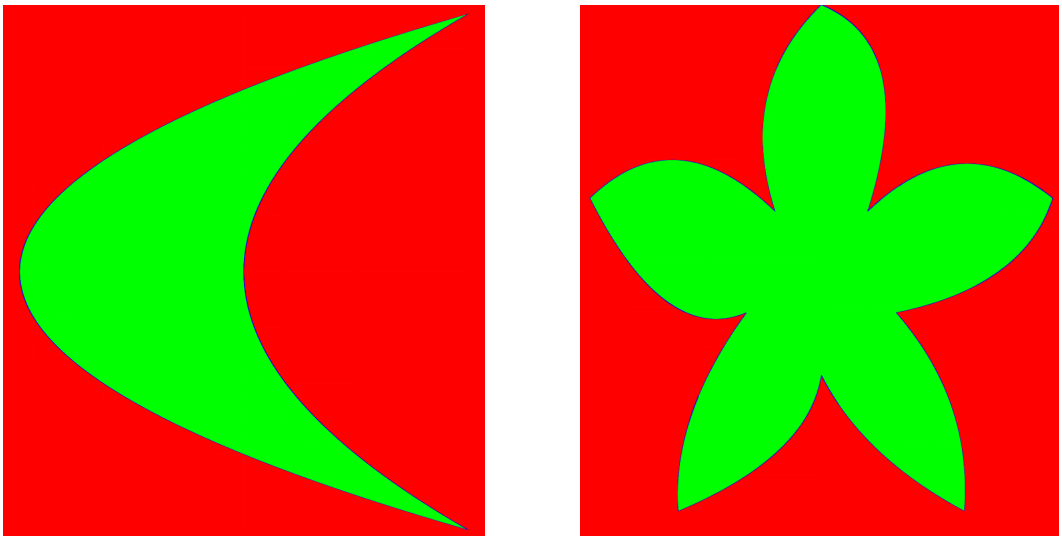


Figure 5.7 – **Objects to pack.** *(Left)* Half–Moon. *(Right)* Flower.

5.6.1 Setup

We have considered 4 experimental cases with 10, 20 and 30 objects to pack each time (in order to test the scalability of the algorithm):

— **Case 1.** Packing of n half-moons, with translation only.

— **Case 2.** Packing of n flowers, with translation only.

— **Case 3.** Mixed packing of $n/2$ half-moons and $n/2$ flowers, with translation only.

— **Case 4.** Packing of n half-moons with rotation.

The description for these objects can be found in the appendix B.

The objects are packed inside a box. The dimension of the box for each case has been set manually so that it can contain n objects but not a lot more.

5.6.2 Paving calculation

The following table presents the paving times of the overlapping regions for all the pair of objects involved in our experimental cases: 2 half moons, 2 flowers, a half moon and a flower, a box and a flower or a half moon, and the pavings of rotated moons.

d.o.f.	*Object i*	*Object j*	*Time*	ε
Translation	Half Moon	Half Moon	1855.44	1
	Flower	Flower	2684.55	3.2
	Flower	Half Moon	10233.17	1
	Enclosing Box	Half Moon	1169.19 (avg)	10
	Enclosing Box	Flower	2472.05 (avg)	10
	Enclosing Box mixed case	Half Moon	1586.32 (avg)	10
	Enclosing Box mixed case	Flower	2074.38 (avg)	10
Translation and Rotation	Half Moon	Half Moon	14400	2
	Enclosing Box	Half Moon	9355.67 (avg)	10

Table 5.1 – **Paving time parametric objects** (in seconds)

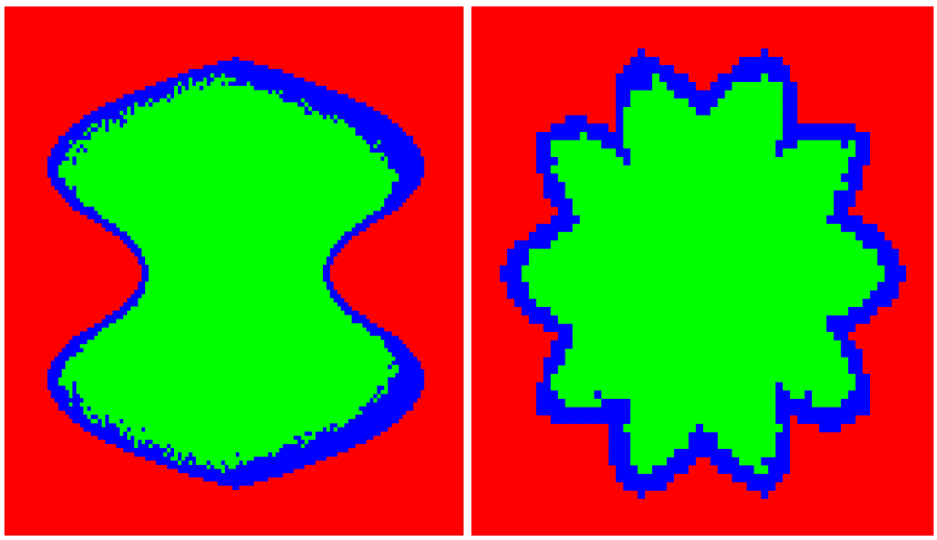


Figure 5.8 – **Overlapping Regions (Left)** Overlapping region between 2 Half Moons. **(Right)** Overlapping region between 2 Flowers.

5.6.3 Packing calculation

The following table presents the packing times. As in the placement of objects described by inequalities, the paving time is very short as compared to the packing times. This means that our approach can probably be used for packing a large number of similar objects.

The quality of the packing solution depends on the precision of the paving of the overlapping region. By looking at the pictures, it seems that the precision has been set correctly as the packing solution makes use of cavities and create interesting near-tangency configurations.

#objects	case 1	case 2	case 3	case 4
10	11.57	22.32	22.27	10.13
20	57.25	108.58	126.80	70.76
30	197.85	228.15	352.66	262.41

Table 5.2 – **Packing time parametric objects** (in seconds)

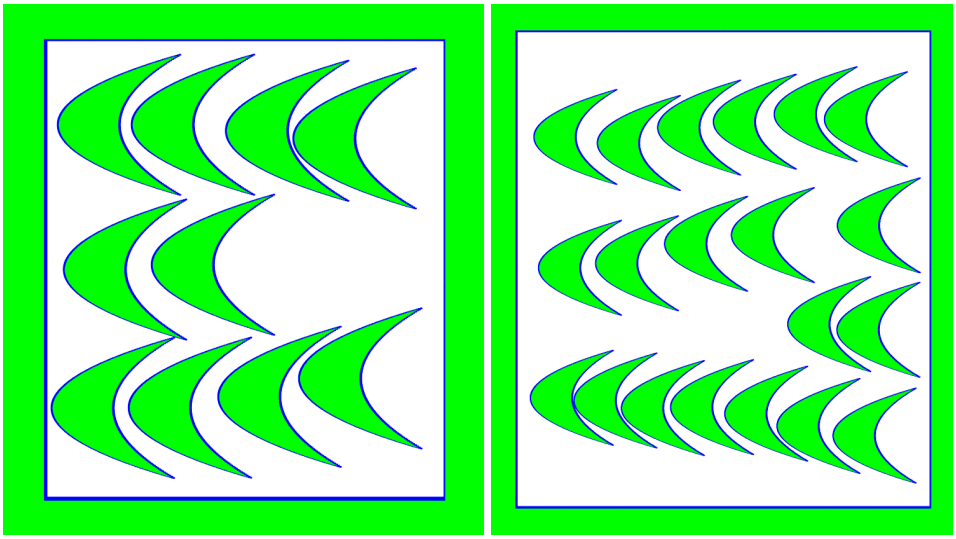


Figure 5.9 – **Packing of 10 and 20 Half Moons. (Left)** Placement of 10 half moons without rotation inside a box. **(Right)** Placement of 20 half moons without rotation inside a box.

5.7 Conclusion

In this chapter, we have introduced a method for paving the overlapping region when the objects are described by parametric curves. The type of curves that are often used in industry are Bézier curves, but our approach is not limited to these. Given that objects are essentially described by connecting pieces of their boundary, the main problem with this representation is the difficulty to deal with the inner and outer regions.

We have first tried to handle the whole problem using a polyhedral approximation of objects. The idea was to compute a set of segments that cover the real object, which can be easily perfomed in the case of Bézier curves using a subdivision method called *de Casteljau's method*. This way, we can calculate the Minkowski sum of the resulting polygons with the help of existing libraries (like the state-of-the-art library CGAL). The main limitation of this approach is the difficulty to calculate the Minkowski sum with orientation.

So, we have proposed to pursue our own approach, by specializing our framework and our two-step inflators to the case of parametric curves. To this end, the following components needed to be revisited: checking the position of a point w.r.t. an object and checking the status of a box w.r.t. an object. Both are based on the fact that a point belongs to an object if and only if a half-line emanating from it plainly crosses

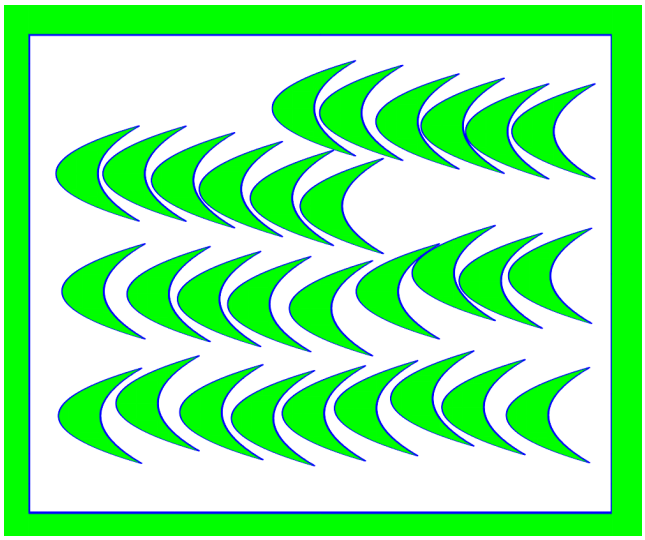


Figure 5.10 – **Packing of 30 Half Moons.** Placement of 30 half moons without rotation inside a box.

the boundary an odd number of times.

Finally, we have presented some preliminar experiments, following a similar setup as in the previous chapter. The results of the experiments show a similar behavior for our framework as with the inequalities. The paving of the overlapping regions can take up to several hours while packing just takes a few minutes. Again, we can observe that the placement of objects takes benefit from the cavaties introduced by the non-convexity of others.

In the case of rotated objects, some implementation issues still need to be fixed; the packing in Pictures 5.15 and 5.16. are due to a very bad precision on the paving boundaries.

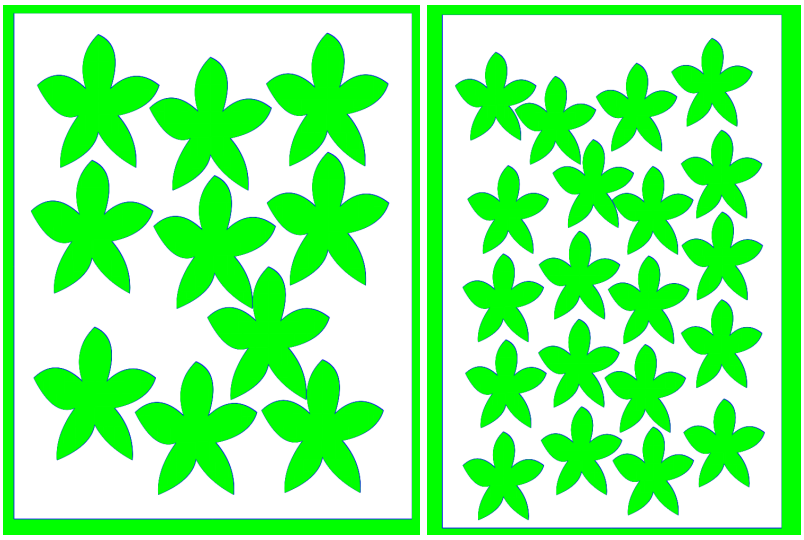


Figure 5.11 – **Packing of 10 and 20 Flowers. (Left)** Placement of 10 flowers without rotation inside a box. **(Right)** Placement of 20 flowers without rotation inside a box.

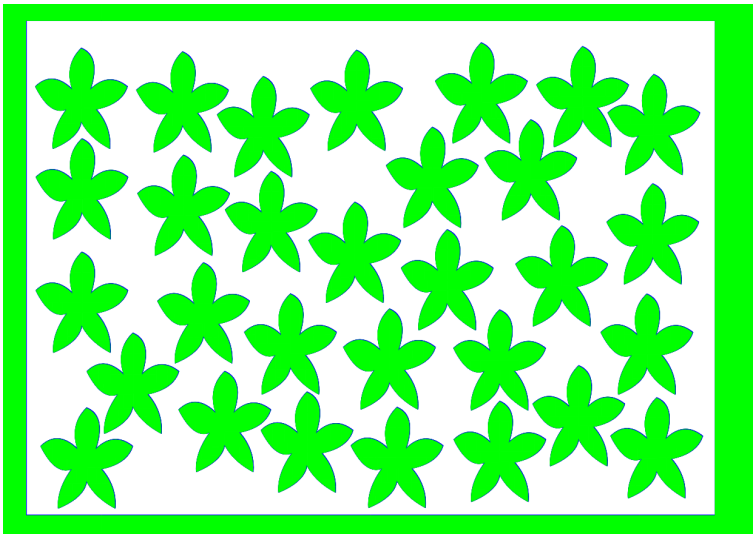


Figure 5.12 – **Packing of 30 Flowers.** Placement of 30 flowers without rotation inside a box.

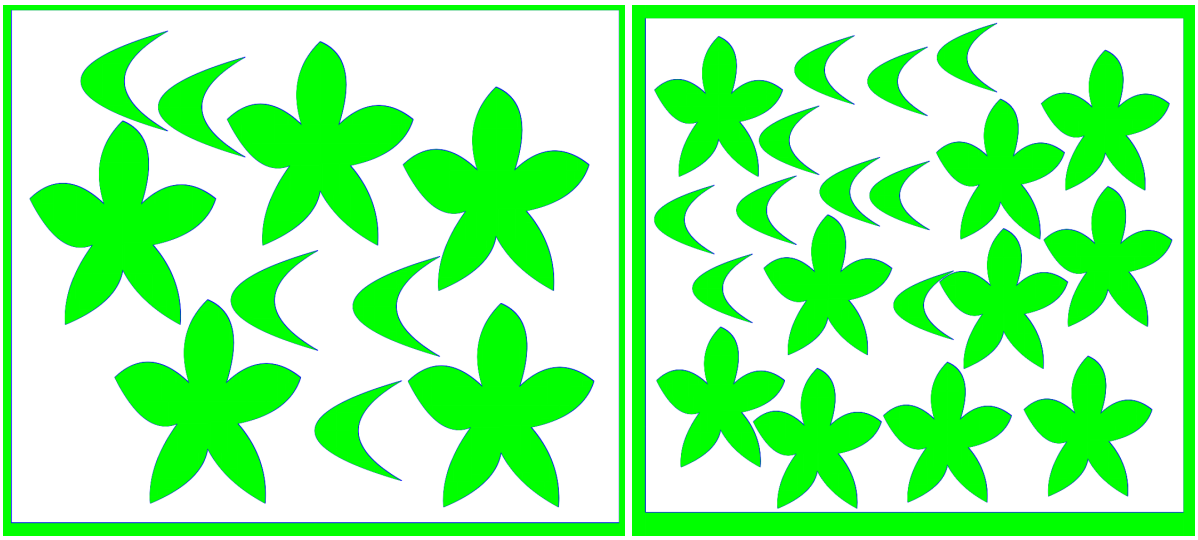


Figure 5.13 – **Mixed Packing of 10 and 20 objects. (Left)** 5 flowers and 5 half moons. **(Right)** 10 flowers and 10 half moons

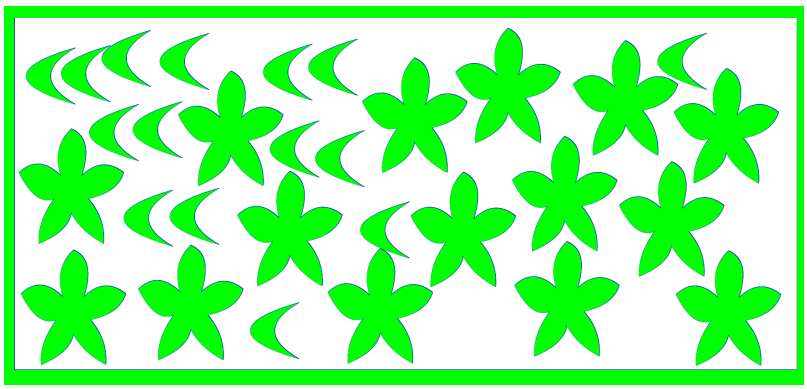


Figure 5.14 – **Mixed Packing of 15 Flowers and 15 Half Moons.**

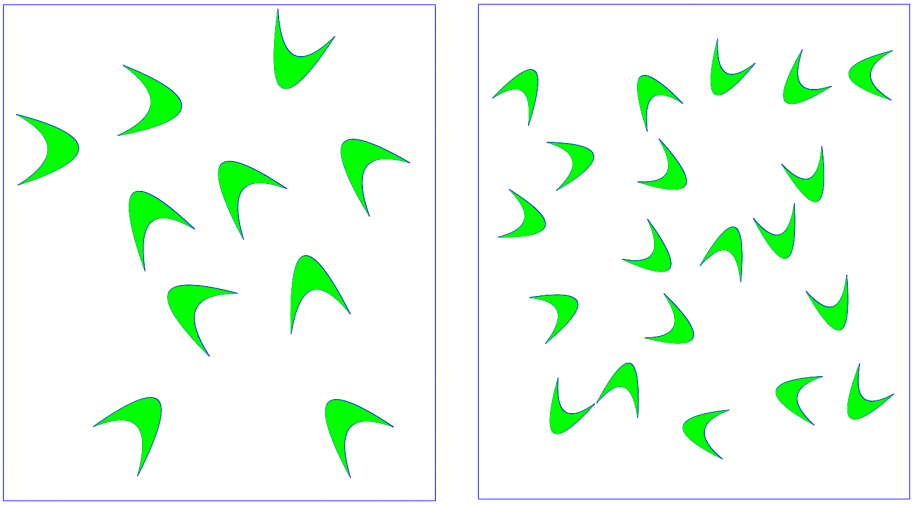


Figure 5.15 – **Packing of 10 and 20 half moons with rotation. (Left)** Placement of 10 Half Moons with rotation inside a box. **(Right)** Placement of 20 Half Moons with rotation inside a box.

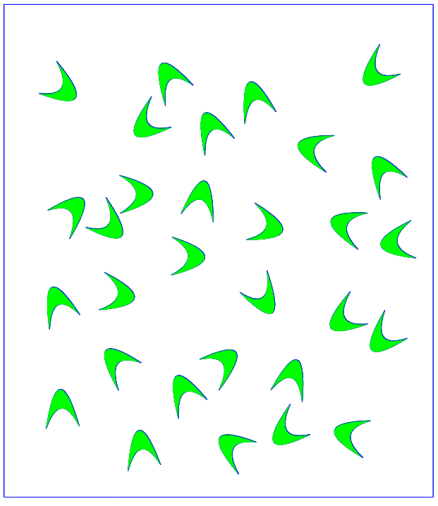


Figure 5.16 – **Packing of 30 half moons.** Placement of 30 Half Moons with rotation inside a box.

6

Conclusion

We have presented in this thesis a new framework for packing curved objects in a container. This framework is generic as it allows a greater freedom in the object definition.

The overall packing algorithm is based on the minimization of an objective function with an evolutionary algorithm. The objective function is an aggregation of so-called *overlapping functions* between all pairs of objects to pack. Finally, an overlapping function is evaluated numerically from the *overlapping region*, a pre-processing of the overlapping constraint. This makes our packing strategy a three-layered algorithm.

The calculation of the overlapping region is performed differently depending how the object is described. Objects can be described in two different ways, either by a logical formula involving inequalities or by a set of parametric curves delimiting its boundary. In both cases, we have proposed a paving algorithm based on an inner test/inflator or outer test/contractor. These operators make use of recent interval techniques, such as the inner arithmetic, and exploit the underlying geometric structure of the problem by separating translation and rotation.

Our approach is not limited to polyhedral shapes and does not make convex approximation. Above all, it does not require ad-hoc formulas for the different types of objects.

The framework has been implemented in C++ and uses the IBEX library for the low-level interval computations.

Preliminary experimental results are encouraging. By observing the graphical results of the packing, we can observe that the cavities introduced by the non-convexity of objects are used to optimize the space. However, some work is still necessary to make the approach scalable, in particular in the case of rotating objects described with parametric curves.

One delicate aspect in the experimental evaluation of our framework is the lack of benchmarks in the literature and, above all, the fact that creating benchmarks is a hard problem on its own: *how should be*

set the dimension of a square so that 30 half-moons fit inside in a compact way? The dimension should neither be minimal or too large so that the packing problem is "difficult" but tractable at the same time. This explains why benchmarks have been created so empirically.

One important continuation of this work is to give the ability to mix objects described by inequalities and parametric curves. This way, the framework would then be fully generic. Such extension should not create insurmountable difficulties. On a more practical side, one possible future experiment could also be to see how both paving and packing time increases as the precision of the pre-processed paving decreases and to mix this numerical approach with ad-hoc formulas, if some are available.

A

Packing more Moons with rotation

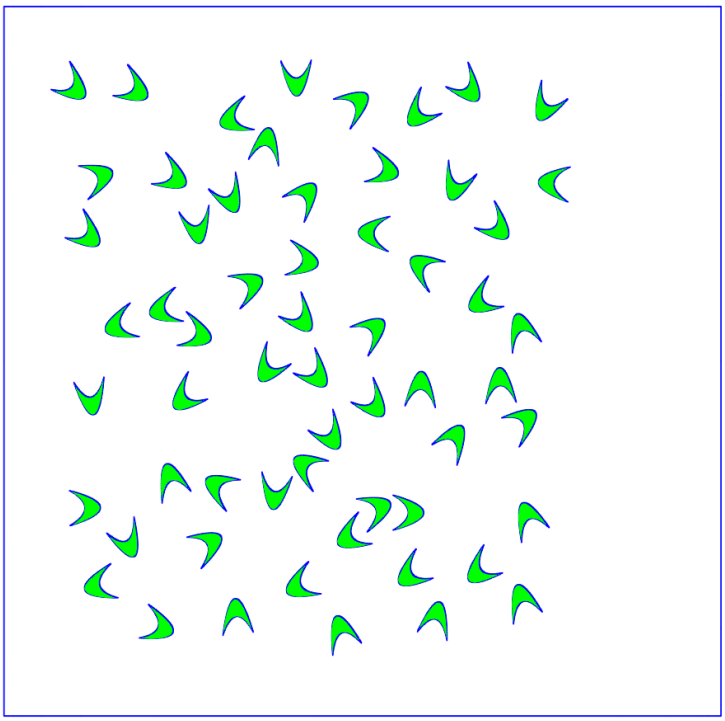


Figure A.1 – **Packing of 60 half moons.** Placement of 60 Half Moons with rotation inside a box.

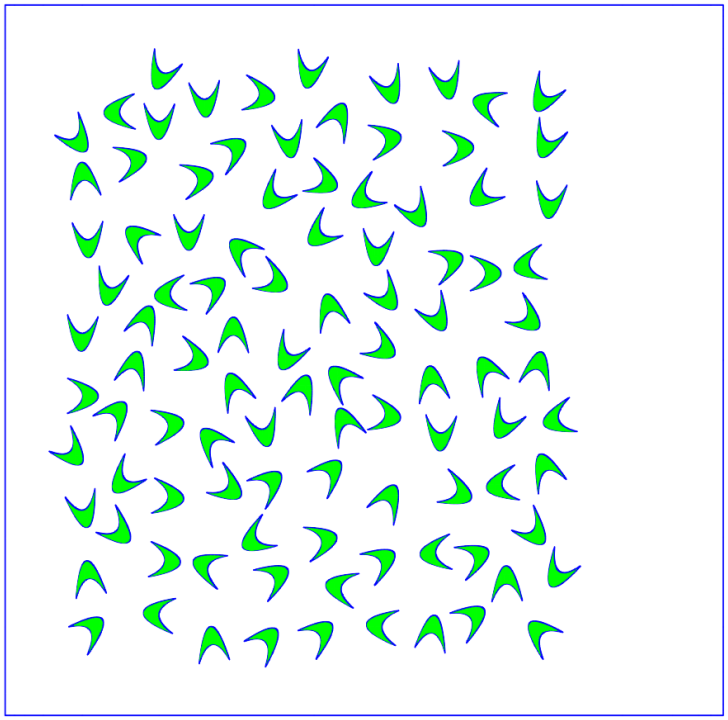


Figure A.2 – **Packing of 100 half moons.** Placement of 100 Half Moons with rotation inside a box.

B

Description of the bezigons to pack in Chapter 5

Control points for each curve of the half moon:

— first curve: $\{(40, 20); (0, 0); (40, -2)\}$
— second curve: $\{(40, 20); (-40, 0); (40, -2)\}$.

Control points for each curve of the flower:

— first curve: $\{(2.2024586, 58.90577); (16.01051, 34.989525); (31.150425, 40.5)\}$
— second curve: $\{(31.150425, 40.5); (17.211597, 23.888353); (18.54966, 8.594238)\}$
— third curve: $\{(18.54966, 8.594238); (42.69821, 17.38359); (45.0, 30.437695)\}$
— fourth curve: $\{(45.0, 30.437695); (52.512943, 17.424889); (71.45033, 8.594238)\}$
— fifth curve: $\{(71.45033, 8.594238); (73.003426, 26.34613); (58.84958, 40.5)\}$
— sixth curve: $\{(58.84958, 40.5); (82.6242, 44.69211); (87.79755, 58.90576)\}$
— seventh curve: $\{(87.79755, 58.90576); (70.514114, 71.00775); (53.55951, 56.78115)\}$
— eighth curve: $\{(53.55951, 56.78115); (63.23457, 83.36317); (45.0, 90.0)\}$
— nineth curve: $\{(45.0, 90.0); (29.181046, 76.72632); (36.44049, 56.781155)\}$
— tenth curve: $\{(36.44049, 56.781155); (18.055468, 72.20802); (2.2024586, 58.90577)\}$

C

Résume

Un problème courant en logistique, gestion d'entrepôt, industrie manufacturière ou gestion d'énergie dans les centres de données est de placer des objets dans un espace limité, ou conteneur. Ce problème est appelé problème de placement. De nombreux travaux dans la littérature gèrent le problème de placement en considérant des objets de formes particulières ou en effectuant des approximations polygonales. Des problèmes typiques sont le *bin packing*, *circle packing*, etc., sachant que, dans un contexte industriel, des contraintes "métier" viennent en général s'ajouter en sus des contraintes géométriques.

Le problème de placement peux également apparaître sous différentes variantes: les objets et le conteneur sont donnés (problème de satisfaction); le conteneur est donné et le nombre d'objets à placer doit être maximisé; les objets sont données et la taille du conteneur doit être minimisé, etc. Dans cette thèse, nous nous intéressons uniquement, par simplicité, au premier cas. Notre approche ne permet pas non plus d'être étendue à l'ajout de contraintes externes "métiers" et traitera le problème de placement d'un seul bloc.

En revanche, notre approche est très générale dans le sens où elle permet de placer des objets de forme arbitraire, dès lors que celle-ci admet une caractérisation mathématique simple, que ce soit avec des inégalités algébriques ou des fonctions paramétrées. En particulier, les objets peuvent être courbes et non-convexes. C'est ce que nous appelons le problème de placement générique.

Le problème de placement générique se place à un niveau important de complexité et représente un champ ouvert de recherche. Nous proposons un algorithme numérique garanti pour le résoudre, en prenant en compte le fait que les objets peuvent être à la fois translatés ou tournés dans un l'espace conteneur.

La description d'un objet nécéssite le choix d'un point particulier de sa forme, appelé *origine*. Cette origine o peut être choisie de façon arbitraire (souvent, le centre de symétrie). Une fois cette origine choisie, la forme de l'objet devient une contrainte au sens habituel du terme, notée $c(p)$, où le point p représente la variable en 2 dimensions décrivant la forme, dans le repère dont l'origine est confondue avec celle de l'objet. Lorsque la contrainte c est limitée à une inégalité (Chapitre 4), la description d'un objet plus complexe est

toujours possible par l'introduction de conjonctions et disjonctions, ex: $c_1 \wedge c_2 \vee c_3$. Ainsi, les primitives géométriques peuvent être assemblées.

La translation d'un objet se traduit alors simplement par le déplacement de son origine, et la contrainte devient donc $c(p - o)$. La rotation de l'objet se traduit également simplement par la rotation inverse appliquée à la variable de la contrainte, c.a.d., $c(R_{-\alpha}(p))$, où α est l'angle de rotation. Les deux peuvent être combinés pour obtenir finalement $c(R_{-\alpha}(p - o))$.

La contrainte centrale est la contrainte de non-chevauchement. Cette contrainte considère deux objets i et j, avec leurs origines o_i et o_j et angles α_i, α_j. Cette contrainte est la négation de la contrainte de chevauchement qui est donc la contrainte existentiellement quantifiée suivante:

$$\exists p \in \mathbb{R}^2, \quad c_i(R_{-\alpha_i}(p - o_i)) \wedge c_j(R_{-\alpha_j}(p - o_j)).$$

L'autre contrainte présente dans le problème de placement générique est la contrainte d'inclusion dans l'objet conteneur. Mais cette contrainte ne nécessite pas en réalité d'être prise en compte, car un objet est inclus dans un conteneur si et seulement s'il ne chevauche pas son complémentaire. Ainsi, le problème consistant à placer n objets $c_1, ..., c_n$ dans un objet conteneur c, est simplement reformulé comme $\binom{n+1}{2}$ contraintes de non-chevauchement entre $n + 1$ objets $c_1, ..., c_n, \neg c$.

Il existe déjà dans la littérature beaucoup des frameworks non dédiés à des objets spécifiques. Ces frameworks ne sont pas aussi généraux que le nôtre mais un certain nombre partagent de fortes similitudes sur le plan méthodologique. Le placement générique est en général effectué en minimisant, via un algorithme d'optimisation adapté, un coût de violation global obtenu par aggrégation de coûts de violation binaire représentant le chevauchement entre deux objets. Les résultats obtenus peuvent être très bons, mais ces approches sont limitées par le recours à des approximations polygonales, ou l'utilisation de fonctions ad-hoc pour mesurer le chevauchement entre objets. Or, ce calcul est une tâche fastidieuse qui limite l'applicabilité de ces approches et leur caractère générique.

Notre contribution pour résoudre le problème de placement est basée sur l'arithmétique d'intervalles. L'arithmétique d'intervalles est une extension de l'arithmétique des réels aux ensembles sous forme d'intervalles. Un intervalle est un sous-ensemble connecté fermé de $\mathbb{R}$. Les intervalles sont notés par des crochets (ex: $[x]$) et l'ensemble des intervalles est noté $\mathbb{IR}$. Un vecteur d'intervalles $[x] \in \mathbb{IR}^d$ est le produit cartésien de d intervalles, $[x] = [x_1] \times ... \times [x_d]$. Nous appelons ces vecteurs d'intervalles *boîtes*.

Les algorithmes-clés basés sur les intervalles et utilisés dans cette thèse sont les suivants: les contracteurs, les inflateurs, l'algorithme de sweep et l'algorithme de pavage. Dans un tout autre registre, nous utilisons également CMA-ES, algorithme évolutionnaire.

Les *contracteurs* sont des algorithmes qui permettent d'enfermer les solutions d'un problème de satisfaction de contraintes en réduisant la taille d'une boîte donnée. La boîte résultant contient l'ensemble des points qui satisfont la contrainte assoicée au contracteur.

Les *inflateurs* sont des opérateurs duaux des contracteurs, qui produisent des boîtes à l'intérieur d'une contrainte. Ils peuvent, pour cela, se baser sur l'arithmétique dite *intérieure*, variante de l'arithmétique d'intervalles qui permet de construire une sous-boîte d'une boîte $[x]$ qui est à l'intérieur d'un ensemble $\mathcal{S}$ décrit par une ou plusieurs inégalités. Cette arithmétique peut être aussi utilisée avec un point initial à partir duquel se fait le gonflement.

L'algorithme de *sweep* est un opérateur produisant un contracteur pour une contrainte à partir d'inflateurs

liés à sa négation. Il produit la contraction d'une boîte en empilant des boîtes produites par les inflateurs jusqu'à ce qu'une face soit complètement couverte.

Enfin, l'algorithme de *pavage* d'un ensemble $\mathcal{S}$ calcule un triplet de boîtes $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ représentant respectivement l'intérieur, la frontière et l'extérieur de l'ensemble $\mathcal{S}$. Cet algorithme est basé sur le découpage récursif d'une boîte initiale, jusqu'à ce qu'il soit possible de classifier chaque sous-boîte dans l'un de ces ensembles, $\mathcal{I}$, $\mathcal{B}$ ou $\mathcal{O}$. Ce découpage peut être maintenu dans une structure arborescente pour en simplifier l'exploration. Deux stratégies différentes de pavage ont été testées dans cette thèse, la stratégie classique *bisect & contract* consistant à bissecter (découper en deux) puis contracter, et la stratégie *bisect & inflate* consistant à laisser le découpage se faire en fonction du résultat du gonflement.

Nous proposons donc un cadre de résolution pour résoudre ce problème de placement générique, basé sur les techniques d'intervalles. Ce cadre possède trois ingrédients essentiels: un algorithme évolutionnaire plaçant les objets, une fonction de chevauchement minimisée par cet algorithme évolutionnaire (coût de violation), et une région de chevauchement qui représente un ensemble pré-calculé des configurations relatives d'un objet (par rapport à un autre) qui créent un chevauchement.

La solution de l'algorithme de placement est une configuration pour chaque objet à placer. Chaque configuration est un vecteur $q_i = (o_i, \alpha_i)$, où o_i est la position de l'objet dans le plan, et α_i l'angle de rotation. Une solution du problème de placement est donc un $d \times n-$tuple $(q_1, ..., q_n)$. Nous modélisons le problème comme un problème d'optimisation continu où la fonction objectif est une combinaison des coûts de violation de chaque contrainte de non-chevauchement $f_{ij}(q_i, q_j)$. Nous avons considéré une simple addition mais d'autre type de combinaisons pourraient être utilisés. La minimisation de cette fonction est externalisée, faite par un algorithme existant (nous avons choisi l'algorithme CMA-ES pour nos expérimentations).

Notre contribution est donc dans la définition et le calcul de la fonction $f_{ij}(q_i, q_j)$, appelé aussi *fonction de chevauchement*. Elle prend donc deux vecteurs de configuration et doit donc donner une mesure du chevauchement, en respectant deux propriétés. D'une part, si les objets sont disjoints, le résultat doit être 0; d'autre part, quand les objets s'éloignent, $f_{ij}(q_i, q_j)$ doit décroître, de telle sorte que la recherche soit guidée vers une séparation des objets. Nous avons alors défini la fonction de chevauchement comme la distance entre le vecteur de configuration initial q_j et le vecteur q'_j le plus proche où le chevauchement n'existe plus entre les objets i et j. Notons que l'objet i reste fixé.

Cette définition embarque un problème de minimisation en 5 dimensions, ce qui signifie que chaque évaluation de f_{ij} coûte trop cher. Notre but est alors d'alléger ce calcul par un pré-traitement de la contrainte de chevauchement. Ce pré-traitement consiste à calculer une région appelée *région de chevauchement*. Cette région représente l'ensemble de configurations pour l'objet j tel que les deux objets se chevauchent mais lorsque la configuration de l'objet i est fixée à 0 (angle inclus).

Cette région de chevauchement est bien sûr calculée distinctement pour chaque paire d'objets. L'algorithme sous-jacent dépend également du fait qu'un objet soit représenté par des inégalités ou des fonctions paramétrées.

Nous montrons alors que la valeur de $f_{ij}(q_i, q_j)$ est égale à la distance minimale entre un vecteur r et l'extérieur de cette région de chevauchement. Ce vecteur r représente simplement la position relative de l'objet j par rapport à l'objet i. Il s'obtient par un calcul direct. Pour calculer la distance avec l'extérieur de la région de chevauchement, nous utilisons le pavage de cette région. Le problème devient un simple calcul de distance entre un point et un ensemble de boîtes (la structure arborescente est exploitée).

Pour calculer la région de chevauchement nous utilisons un inflateur intérieur et un test de rejet extérieur (en guise de contracteur). L'inflateur intérieur que nous proposons comporte deux étapes: une inflation par rapport à la translation, et une inflation par rapport à l'orientation. L'inflation par rapport à la translation repose sur l'équivalence entre le problème sans orientation et le calcul d'une somme de Minkowski. Le principe consiste à construire une boîte à l'intérieur de l'objet i à partir d'un point $\tilde{p}$ à l'intérieur de l'objet j. Cette boîte peut alors être translatée autour de l'origine de l'objet j. Il serait possible également d'y ajouter une boîte à l'intérieur de l'objet j, mais cet ajout supplémentaire ne serait plus compatible avec la seconde étape. La seconde étape consiste donc en une inflation par rapport à l'orientation. L'idée consiste à calculer les deux valeurs d'angle les plus proche de l'angle d'origine telles que le point $\tilde{p}$ rencontre la frontière de l'objet j. Or, ce point $\tilde{p}$ appartient à l'objet i quelles que soient les positions de l'origine obtenues via la première étape d'inflation. En conséquence, si le résutlat de la première étape est un rectangle $[o_j]$ et le résultat de la seconde étape un intervalle $[\underline{\alpha_j}, \overline{\alpha_j}]$, l'inflation finale est simplement le produit cartésien $[o_j] \times [\underline{\alpha_j}, \overline{\alpha_j}]$.

Dans le cas d'objets décrits par des inégaliés, la construction d'une boîte à l'intérieur d'un objet ne pose pas de problème particulier: il suffit d'utiliser l'arithmétique intérieure (technique assez récente publiée par un des co-encadrants de cette thèse). Le cas des objets dérits par des courbes paramétrés est plus complexe. Du fait que ces objets soient essentiellement décrit par des raccolements de morceaux de courbes, la propriété mathématique d'être à l'intérieur ou à l'extérieur est beaucoup plus indirecte et ne se généralise pas immédiatement pour en faire un inflateur.

Par ailleurs, les courbes paramétriques se prêtent particulièrement bien à l'approximation polyhédrale. C'est encore plus vrai pour les courbes de Bézier, ou l'algorithme de De Casteljau permet aisément d'obtenir une telle approximation, sans avoir recours à des linéarisations (vecteurs dérivées) ni à l'arithmétique d'intervalles (pour une approximation garantie). Sachant qu'une somme de Minkowski est facile à calculer avec des polyhèdres, et que des librairies telles que CGAL en offre une implémentation efficace, l'approximation polyhédrale est la voie royale pour traiter le problème de placement générique avec des courbes paramétrées. Néanmoins, cela se limite au cas de la translation.

Dans la perspective de généraliser notre framework au cas des courbes paramétrées, nous avons donc élaboré une technique pour tester si une boîte est intérieure, à défaut de pouvoir proposer un infalteur. Nous montrons que cela suffit pour obtenir un algorithme de pavage, au prix sans doute d'une certaine perte d'efficacité. Notre technique repose elle-même sur un algorithme testant si un point est à l'intérieur d'un objet, algorithme obtenu en appliquant le célèbre théorème de Jordan pour les courbes planes, à savoir, qu'un point appartient à un objet si et seulement si une demi-droite émanant de celui-ci intersecte pleinement (c.a.d. de façon non-tangentielle) la frontière un nombre impair de fois.

Des expérimentations préliminaires pour les deux catégories d'objets permettent de valider notre approche et d'en montrer le potentiel. Comme attendu, le calcul du pavage des régions de chevauchement pour chaque paire d'objet peut prendre un temps considérable (plusieurs heurs), mais ces pavages ne sont calculés qu'une seule fois quel que soit le nombre d'objets placés et peuvent être partagés par plusieurs instances de placement générique ayant des formes à placer en commun. De plus, une comparaison avec un logiciel de l'état de l'art sur la résolution de contraintes quantifiées montre que notre technique pour le calcul de pavages est, en fait, compétitive.

Nous montrons que nous pouvons placer une trentaine d'objects en un temps de l'ordre de la minute (de plus larges instances n'ont pas été étudiées). Il peut être observé graphiquement que le placement produit

des configurations intéressantes où la non-convexité permet aux objets d'être “imbriqués”.

Contents

List of Tables

List of Figures

Bibliography

[Araya 2012] I. Araya, G. Trombettoni et B. Neveu. *A Contractor Based on Convex Interval Taylor*. In CPAIOR, volume 7298, pages 1–16, 2012. 22

[Araya 2014] I. Araya, G. Trombettoni, B. Neveu et G. Chabert. *Upper Bounding in Inner Regions for Global Optimization Under Inequality Constraints*. Journal of Global Optimization, vol. 60, no. 2, pages 145–164, 2014. 22, 28

[Beldiceanu 2001] Nicolas Beldiceanu et Mats Carlsson. *Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint*. In Principles and Practice of Constraint Programming-CP 2001, pages 377–391. Springer, 2001. 24, 33

[Benhamou 1999] F. Benhamou, F. Goualard, L. Granvilliers et J-F. Puget. *Revising Hull and Box Consistency*. In ICLP, pages 230–244, 1999. 22

[Benhamou 2006] F. Benhamou et L. Granvilliers. *Continuous and Interval Constraints*. In Handbook of Constraint Programming, chapitre 16, pages 571–604. Elsevier, 2006. 22

[Boehm 1999] Wolfgang Boehm et Andreas Müller. *On de Casteljau's algorithm*. Computer Aided Geometric Design, vol. 16, no. 7, pages 587–605, 1999. 67

[Brochu 2012] T. Brochu, E. Edwards et R. Bridson. *Efficient Geometrically Exact Continuous Collision Detection*. ACM Transactions on Graphics (TOG), vol. 31, no. 4, page 96, 2012. 14

[Broyden 1970] Charles George Broyden. *The convergence of a class of double-rank minimization algorithms 1. general considerations*. IMA Journal of Applied Mathematics, vol. 6, no. 1, pages 76–90, 1970. 13

[Cambazard 2013] Hadrien Cambazard, Deepak Mehta, Barry O'Sullivan et Helmut Simonis. *Bin packing with linear usage costs–an application to energy management in data centres*. In Principles and Practice of Constraint Programming, pages 47–62. Springer, 2013. 5

[Cameron 1986] S. Cameron et R. Culley. *Determining the Minimum Translational Distance Between Two Convex Polyhedra*. In IEEE International Conference on Robotics and Automation, volume 3, pages 591–596. IEEE, 1986. 42

[Carnicer 1993] Jesús M Carnicer et Juan Manuel Peña. *Shape preserving representations and optimality of the Bernstein basis*. Advances in Computational Mathematics, vol. 1, no. 2, pages 173–196, 1993. 66

[Chabert 2009] G. Chabert et L. Jaulin. *Contractor Programming*. Artificial Intelligence, vol. 173, no. 11, pages 1079–1100, 2009. 21, 22, 26

[Chabert 2010] G. Chabert et N. Beldiceanu. *Sweeping with Continous Domains*. In CP, International Conference on Principles and Practice of Constraint Programming, volume 6308 of *LNCS*, pages 137–151, 2010. 22, 24

[Cheng 1994] CH Cheng, BR Feiring et TCE Cheng. *The cutting stock problem-a survey*. International Journal of Production Economics, vol. 36, no. 3, pages 291–305, 1994. 5

[Chernov 2010] N Chernov, Yu Stoyan et Tatiana Romanova. *Mathematical model and efficient algorithms for object packing problem*. Computational Geometry, vol. 43, no. 5, pages 535–553, 2010. 11

[Cohen 1995] Jonathan D Cohen, Ming C Lin, Dinesh Manocha et Madhav Ponamgi. *I-collide: An interactive and exact collision detection system for large-scale environments*. In Proceedings of the 1995 symposium on Interactive 3D graphics, pages 189–ff. ACM, 1995. 14

[Cui 2013] Yaodong Cui, Liu Yang, Zhigang Zhao, Tianbing Tang et Mengxiao Yin. *Sequential grouping heuristic for the two-dimensional cutting stock problem with pattern reduction*. International Journal of Production Economics, vol. 144, no. 2, pages 432–439, 2013. 5

[Deb 2008] Kalyanmoy Deb et Santosh Tiwari. *Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization*. European Journal of Operational Research, vol. 185, no. 3, pages 1062–1087, 2008. 13

[Dobkin 1993] D. Dobkin, J. Hershberger, D. Kirkpatrick et S. Suri. *Computing the Intersection-Depth of Polyhedra*. Algorithmica, vol. 9, no. 6, pages 518–533, 1993. 42

[Domovic 2014] D Domovic, T Rolich, D Grundler et S Bogovic. *Algorithms for 2D nesting problem based on the no-fit polygon*. In Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on, pages 1094–1099. IEEE, 2014. 13

[Drira 2007] Amine Drira, Henri Pierreval et Sonia Hajri-Gabouj. *Facility layout problems: A survey*. Annual Reviews in Control, vol. 31, no. 2, pages 255–267, 2007. 5

[Fabri 1998] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra et Sven Schönherr. *On the design of CGAL, the computational geometry algorithms library*. 1998. 68

[Farouki 2012] Rida T Farouki. *The Bernstein polynomial basis: a centennial retrospective*. Computer Aided Geometric Design, vol. 29, no. 6, pages 379–419, 2012. 66

[Goldsztejn 2006] A. Goldsztejn et L. Jaulin. *Inner and Outer Approximations of Existentially Quantified Equality Constraints*. In CP, pages 198–212. Springer, 2006. 15

[Gomes 2006] A Miguel Gomes et José F Oliveira. *Solving irregular strip packing problems by hybridising simulated annealing and linear programming*. European Journal of Operational Research, vol. 171, no. 3, pages 811–829, 2006. 13

[Hansen 1978] E.R. Hansen. *Interval Forms of Newton's Method*. Computing, vol. 20, pages 153–163, 1978. 22

[Hansen 1992] E.R. Hansen. Global Optimization using Interval Analysis. Marcel Dekker, 1992. 28

[Hansen 2001] N. Hansen et A. Ostermeier. *Completely Derandomized Self-Adaptation in Evolution Strategies*. Evolutionary Computation, vol. 9, no. 2, pages 159–195, 2001. 12, 28

[Hifi 2009] Mhand Hifi et Rym M'hallah. *A literature review on circle and sphere packing problems: models and methodologies*. Advances in Operations Research, vol. 2009, 2009. 5

[Ishii 2012] D. Ishii, A. Goldsztejn et C. Jermann. *Interval-Based Projection Method for Under-Constrained Numerical Systems*. Constraints, vol. 17, no. 4, pages 432–460, 2012. 15

[Jacquenot 2010] Guillaume Jacquenot. *Méthode générique pour l'optimisation d'agencement géométrique et fonctionnel*. PhD thesis, Ecole Centrale de Nantes (ECN), 2010. 13

[Jain 2012] Megha Jain, Pranjali Arondekar et Ashok Ganguly. *A JOURNEY FROM DE-BOOR'S ALGORITHM TO DE CASTELJAU'S ALGORITHM FOR THE CONSTRUCTION OF B-SPLINE SURFACES & NURBS*. International Journal of Mathematical Archive (IJMA) ISSN 2229-5046, vol. 3, no. 11, 2012. 67

[Jaulin 2001] Luc Jaulin. Applied interval analysis: with examples in parameter and state estimation, robust control and robotics, volume 1. Springer Science & Business Media, 2001. 20, 25, 26

[Jolly 2009] KG Jolly, R Sreerama Kumar et R Vijayakumar. *A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits*. Robotics and Autonomous Systems, vol. 57, no. 1, pages 23–33, 2009. 65

[Jouida 2015] Sihem Ben Jouida, Ahlem Ouni et Saoussen Krichen. *A Multi-start Tabu Search Based Algorithm for Solving the Warehousing Problem with Conflict*. In Modelling, Computation and Optimization in Information Systems and Management Sciences, pages 117–128. Springer, 2015. 5

[Kim 2002] Y.J. Kim, M.A. Otaduy, M.C. Lin et D. Manocha. *Fast Penetration Depth Computation for Physically-Based Animation*. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 23–31. ACM, 2002. 42

[Kim 2004] Y.J. Kim, M.C. Lin et D. Manocha. *Incremental Penetration Depth Estimation Between Convex Polytopes Using Dual-Space Expansion*. IEEE Transactions on Visualization and Computer Graphics, vol. 10, no. 2, pages 152–163, 2004. 42

[Kirk 2012] David B Kirk et W Hwu Wen-mei. Programming massively parallel processors: a hands-on approach. Newnes, 2012. 65

[Lebbah 2004] Y. Lebbah, C. Michel, M. Rueher, D. Daney et J-P. Merlet. *Efficient and Safe Global Constraints for handling Numerical Constraint Systems*. SIAM J. Numer. Anal., vol. 42, no. 2, pages 505–529, 2004. 22

[Lhomme 1993] O. Lhomme. *Consistency Techniques for Numeric CSPs*. In IJCAI, pages 232–238, 1993. 22

[Lin 1993] Ming C Lin. *Efficient collision detection for animation and robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993. 14

[Lin 2015] Chin-Jung Lin, Maw-Sheng Chern et Mingchang Chih. *A binary particle swarm optimization based on the surrogate information with proportional acceleration coefficients for the 0-1 multidimensional knapsack problem*. Journal of Industrial and Production Engineering, pages 1–26, 2015. 5

[Lodi 2002] A. Lodi, S. Martello et M. Monaci. *Two-dimensional Packing Problems: A Survey*. European Journal of Operational Research, vol. 141, no. 2, pages 241–252, 2002. 5

[Madhumathi 2015] R Madhumathi, R Radhakrishnan et AS Balagopalan. *Dynamic resource allocation in cloud using bin-packing technique*. In Advanced Computing and Communication Systems, 2015 International Conference on, pages 1–4. IEEE, 2015. 5

[Mainzer 2015] D. Mainzer et G. Zachmann. *Collision Detection based on Fuzzy Scene Subdivision*. In GPU Computing and Applications, pages 135–150. Springer, 2015. 14

[Martinez 2013] Thierry Martinez, Lumadaiara Vitorino, François Fages et Abderrahmane Aggoun. *On Solving Mixed Shapes Packing Problems by Continuous Optimization with the CMA Evolution*

Strategy. In Computational Intelligence and 11th Brazilian Congress on Computational Intelligence (BRICS-CCI & CBIC), 2013 BRICS Congress on, pages 515–521. IEEE, 2013. 12, 60

[Mistry 2014] Shanaz Mistry, UN Niranjan et M Gopi. *Puzzhull: Cavity and protrusion hierarchy to fit conformal polygons*. Computer-Aided Design, vol. 46, pages 233–238, 2014. 13

[Moore 1966] R. Moore. Interval Analysis. Prentice-Hall, 1966. 20

[Moore 1988] M. Moore et J. Wilhelms. *Collision Detection and Response for Computer Animation*. ACM Siggraph Computer Graphics, vol. 22, no. 4, pages 289–298, 1988. 14

[Neumaier 1990] A. Neumaier. Interval Methods for Systems of Equations. Cambridge University Press, 1990. 22

[Neveu 2015] B. Neveu, G. Trombettoni et I. Araya. *Adaptive Constructive Interval Disjunction: Algorithms and Experiments*. Constraints, vol. 20, no. 4, pages 452–467, 2015. 22

[Pisinger 2007] David Pisinger. *The quadratic knapsack problem-a survey*. Discrete applied mathematics, vol. 155, no. 5, pages 623–648, 2007. 5

[Pospichal 2015] Jiri Pospichal. *Solving Circle Packing Problem by Neural Gas*. In Mendel 2015, pages 281–289. Springer, 2015. 5

[Ratschan 2006a] S. Ratschan. *Efficient Solving of Quantified Inequality Constraints over the Real Numbers*. ACM Transactions on Computational Logic, vol. 7, no. 4, pages 723–748, 2006. 15

[Ratschan 2006b] S. Ratschan. *RSolver*, 2006. 15

[Ross 1996] Sheldon M Ross*et al.* Stochastic processes, volume 2. John Wiley & Sons New York, 1996. 29

[Salas 2014] I. Salas, G. Chabert et A. Goldsztejn. *The Non-Overlapping Constraint between Objects described by Non-Linear Inequalities*. In CP, International Conference on Principles and Practice of Constraint Programming, volume 8656 of *LNCS*, pages 672–687, 2014. 33

[Sam-Haroud 1996] D. Sam-Haroud et B. Faltings. *Consistency Techniques for Continuous Constraints*. Constraints, vol. 1, pages 85–118, 1996. 26

[Sato 2012] André Kubagawa Sato, Thiago Castro Martins et Marcos Sales Guerra Tsuzuki. *An algorithm for the strip packing problem using collision free region and exact fitting placement*. Computer-Aided Design, vol. 44, no. 8, pages 766–777, 2012. 13

[Shah 1995] Jami J Shah et Martti Mäntylä. Parametric and feature-based cad/cam: concepts, techniques, and applications. John Wiley & Sons, 1995. 65

[Snyder 2014] John M Snyder. Generative modeling for computer graphics and cad: symbolic shape design using interval analysis. Academic press, 2014. 65

[Stoyan 2013] Yuri Stoyan et Tatiana Romanova. *Mathematical models of placement optimisation: two- and three-dimensional problems and applications*. In Modeling and Optimization in Space Engineering, pages 363–388. Springer, 2013. 11

[Tănase 2004] Mirela Tănase et Remco C Veltkamp. *A straight skeleton approximating the medial axis*. In Algorithms–ESA 2004, pages 809–821. Springer, 2004. 13

[Tari 2015] Farhad Ghassemi Tari et Hossein Neghabi. *A new linear adjacency approach for facility layout problem with unequal area departments*. Journal of Manufacturing Systems, vol. 37, pages 93–103, 2015. 5

[Trombettoni 2007] G. Trombettoni et G. Chabert. *Constructive Interval Disjunction*. In Principles and Practice of Constraint Programming, pages 635–650, 2007. 22

[Wolfson 1988] Haim Wolfson, Edith Schonberg, Alan Kalvin et Yehezkel Lamdan. *Solving jigsaw puzzles by computer*. Annals of Operations Research, vol. 12, no. 1, pages 51–64, 1988. 13

[Zachariadis 2012] Emmanouil E Zachariadis, Christos D Tarantilis et Chris T Kiranoudis. *The pallet-packing vehicle routing problem*. Transportation Science, vol. 46, no. 3, pages 341–358, 2012. 5

[Zhang 2013] Yunong Zhang, Bingguo Mu et Huicheng Zheng. *Link between and comparison and combination of Zhang neural network and quasi-Newton BFGS method for time-varying quadratic minimization*. Cybernetics, IEEE Transactions on, vol. 43, no. 2, pages 490–503, 2013. 13

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

MINES Nantes
Institut Mines-Télécom

Thèse de Doctorat

Ignacio Antonio SALAS DONOSO

Méthodes Intervalles pour le Placement d'Objets Courbes

Packing Curved Objects with Interval Methods

Résumé

Un problème courant en logistique, gestion d'entrepôt, industrie manufacturière ou gestion d'énergie dans les centres de données est de placer des objets dans un espace limité, ou conteneur. Ce problème est appelé problème de placement. De nombreux travaux dans la littérature gèrent le problème de placement en considérant des objets de formes particulières ou en effectuant des approximations polygonales. L'objectif de cette thèse est d'autoriser toute forme qui admet une définition mathématique (que ce soit avec des inégalités algébriques ou des fonctions paramétrées). Les objets peuvent notamment être courbes et non-convexes. C'est ce que nous appelons le problème de placement générique. Nous proposons un cadre de résolution pour résoudre ce problème de placement générique, basé sur les techniques d'intervalles. Ce cadre possède trois ingrédients essentiels : un algorithme évolutionnaire plaçant les objets, une fonction de chevauchement minimisée par cet algorithme évolutionnaire (coût de violation), et une région de chevauchement qui représente un ensemble pré-calculé des configurations relatives d'un objet (par rapport à un autre) qui créent un chevauchement. Cette région de chevauchement est calculée de façon numérique et distinctement pour chaque paire d'objets. L'algorithme sous-jacent dépend également du fait qu'un objet soit représenté par des inégalités ou des fonctions paramétrées. Des expérimentations préliminaires permettent de valider l'approche et d'en montrer le potentiel.

Mots clés

Problème de placement, Contrainte de non-chevauchement, Somme de Minkowski, Arithmétique d'Intervalles, Courbes paramétrées, Pavage.

Abstract

A common problem in logistic, warehousing, industrial manufacture, newspaper paging or energy management in data centers is to allocate items in a given enclosing space or container. This is called a packing problem. Many works in the literature handle the packing problem by considering specific shapes or using polygonal approximations. The goal of this thesis is to allow arbitrary shapes, as long as they can be described mathematically (by an algebraic equation or a parametric function). In particular, the shapes can be curved and non-convex. This is what we call the generic packing problem. We propose a framework for solving this generic packing problem, based on interval techniques. The main ingredients of this framework are: An evolutionary algorithm to place the objects, an overlapping function to be minimized by the evolutionary algorithm (violation cost), and an overlapping region that represents a pre-calculated set of all the relative configurations of one object (with respect to the other one) that creates an overlapping. This overlapping region is calculated numerically and distinctly for each pair of objects. The underlying algorithm also depends whether objects are described by inequalities or parametric curves. Preliminary experiments validate the approach and show the potential of this framework.

Key Words

Packing problem, Non-overlapping Constraint, Minkowski sum, Interval Arithmetic, Parametric curves, Paving.